Distributed Inference Using Oscillatory Dynamics in Neural Networks and Neurally-inspired VLSI Systems

Dissertation zur Erlangung der naturwissenschaftlichen Doktorwürde

(Dr. sc. nat.)

vorgelegt der

Mathematisch-naturwissenschaftlichen Fakultät

der

Universität Zürich

von

Hesham Mostafa Elsayed

von

Ägypten

Promotionskommittee

Prof. Dr. Giacomo Indiveri (Vorsitz)

Prof. Dr. Tobi Delbruck

Prof. Dr. Richard Hahnloser

Zürich 2016

Acknowledgments

This thesis would not have been possible without the help and support of many people. I would like to thank my supervisor Giacomo Indiveri for his support, and for giving me the freedom and time to pursue the ideas and research goals that I find interesting. I am grateful to Lorenz Muller for many deep and interesting discussions and an intellectually stimulating collaboration. I would like to thank the people of INI for a wonderful work environment. My deepest gratitude is for my family who unobtrusively imparted to me the view that doing a PhD is a natural step in life, I appreciate what you did.

Abstract

Biological neural networks process information in an executive-free manner with no central controller managing the flow of information and no strict separation between computation and memory primitives. This mode of computation exhibits superior performance in many biologically relevant computational problems, such as visual perception and motor control, compared to conventional von Neumann architectures operating within the same power budget. Biological networks thus provide a tantalizing proof of the existence of a physically implementable computing architecture that is distributed, fault-tolerant, and adaptive and that outperforms conventional architectures in many important problems. To engineer a biologically-inspired system that reproduces these features, two main questions have to be addressed: how does efficient computation arise from the collective dynamics of biologically-inspired networks? and how to implement the computationally relevant aspects of these dynamics on a physical substrate such as Very Large Scale Integration (VLSI) chips? The two parts of this thesis deal with these two questions, primarily as they relate to the difficult problems of inference and constraint satisfaction.

The first part of this thesis investigates how usable computation can be obtained from the dynamics of recurrently connected neural networks. I start by investigating the dynamics of coupled attractor networks and show that an anatomically justified asymmetric coupling scheme allows these networks to exhibit stable and input-dependent patterns of sequential activity. I show that these sequential activity patterns can be used to implement self-timed symbolic computation where input symbols are combined and successively transformed in a state-dependent way through multiple stages to produce output symbols. Asymmetrically coupled attractor networks could thus be used to implement finite state machines. I then investigate the dynamics of oscillatory coupled attractor networks and argue that Gamma-band rhythmic inhibition can allow these networks to search for maximally consistent interpretations of ambiguous or incomplete inputs. These oscillatory networks could thus be used as a biologically motivated neural substrate on which to ground various "perception as inference" theories. They also highlight a new way of solving constraint satisfaction problems.

The second part of the thesis deals with the construction of neuromorphic systems and the physical implementation of neurally-inspired algorithms. I begin by describing the address event representation (AER) protocol for event communication and present an automated methodology for synthesizing AER interface circuits from behavioral VHDL code. These automatically synthesized AER interface circuits are part of the necessary communication infrastructure of the event-based computing architecture that I developed. I then present a hybrid CMOS-memristor architecture that uses memristors as plastic synaptic elements and present measurements from a physical implementation of this architecture. This is the first time that memristors are directly interfaced to a neuromorphic multi-neuron chip and through this implementation, I explore the opportunities and challenges involved in using memristors to address the synaptic scaling bottleneck of neuromorphic systems. I then describe the *inferenceEngine* architecture which is a

distributed, event-based architecture for carrying out inference operations and solving constraint satisfaction problems. This architecture represents a more efficient alternative for solving these important problems compared to conventional von Neumann architectures and is inspired by the oscillatory neural networks presented in the first part of the thesis. I describe how various problems such as Boolean satisfiability, graph coloring, and the traveling salesman problem can be solved by this architecture. Even though the *inferenceEngine* architecture is fully deterministic, I develop a stochastic interpretation of its dynamics that accurately predicts the statistics of these dynamics. I use this stochastic interpretation to show that stochastic networks such as restricted Boltzmann machines can be implemented on this architecture. I present measurements from a physical VLSI realization of the *inferenceEngine* architecture. In the concluding section, I discuss the relevance of the work done and directions for future work.

Some of the most difficult and practically useful problems in the fields of computer science, optimization, and machine learning can be formulated as "best-match" problems where the goal is to find the values of a large number of variables that maximally satisfy a set of constraints. There is a fundamental relation between these problems and the relaxation to minimum energy states in several physical stochastic systems. The main contribution of this work is establishing another fundamental relation between "best-match" problems and the dynamics of oscillatory systems. I show that event-based systems of coupled VLSI oscillators that exploit these novel dynamics have a clear potential to outperform existing von-Neumann computing architectures in terms of speed and power consumption when solving various types of "best-match" problems, provided they employ an efficient and distributed event-routing scheme. For example, simulations indicate that the proposed VLSI architecture can solve instances of the widely applicable boolean satisfiability problem three orders of magnitude faster than a conventional computer running at the same average frequency. In addition to their technological potential, the novel dynamics used to solve "best-match" problems motivate a novel non-stochastic formulation of "perceptual inference" phenomena that relate these phenomena to cortical rhythms instead of neural or synaptic noise.

Zusammenfassung

Biologische Neuronale Netze verarbeiten Informationen auf eine “executive-free” Art, ohne zentrale Steuereinheit zur Verwaltung des Informationsflusses und ohne strikte Trennung von Rechen- und Speicher Primitiven. Verglichen mit herkömmlichen von-Neumann-Architekturen zeigt diese Art der Informationsverarbeitung eine überlegene Leistung in vielen biologisch relevanten Berechnungsproblemen, wie die visuelle Wahrnehmung und Bewegungssteuerung, bei gleichbleibendem Leistungsbudget. Biologische Netzwerke bieten somit einen verlockende Beweis für die Existenz einer physikalisch realisierbaren verteilten Rechenarchitektur, die fehlertolerant und adaptiv ist und gegenüber herkömmlichen Architekturen in vielen wichtigen Problemen schlägt. Um ein biologisch-inspiriertes System welches diese Eigenschaften wiederspiegelt zu konstruieren, muessen zwei Fragestellungen behandelt werden: Wie entsteht effiziente Problemloesung aus der kollektiven Dynamik biologisch inspirierter Netzwerke? und wie implementiert man die bei dieser Berechnung systemrelevanten aspekte auf einem physischen Substrat, wie etwa Very Large Scale Integration (VLSI) Chips? Die beiden Teile dieser Arbeit befassen sich mit den genannten Fragestellungen, vor allem im Bezug auf Inferenz- und Constraint Satisfaction Probleme.

Der erste Teil dieser Arbeit untersucht, wie brauchbare Informationsverarbeitung mithilfe der Dynamiken rekurrenter Netze erreicht werden. Ich beginne mit der Untersuchung der Dynamik gekoppelter Attraktornetzwerke und zeige, dass ein anatomisch begründetes asymmetrisches Kopplungsschema es ermöglicht, diese Netzwerke stabile und Eingabeabhängigen Muster sequenzieller Aktivität aufweisen zu lassen. Weiterhin zeige ich, dass diese sequenziellen Aktivitätsmuster genutzt werden können um selbstgetaktete symbolische Berechnungen zu implementieren, in denen Symbole kombiniert und nacheinander in einer zustandsabhängigen Weise transformiert werden um Ausgabesymbole zu generieren. Asymmetrisch gekoppelt Attraktornetze könnten daher verwendet werden, um Zustandsautomaten zu implementieren. Weiterhin untersuche ich die Dynamik von oszillierenden gekoppelten Attraktor Netzwerken und argumentiere, dass die rhythmische Inhibition durch das Gamma-Band es zulässt, mittels dieser Netze nach maximal konsistenten Interpretationen von mehrdeutigen oder unvollständige Eingaben zu suchen. Diese Oszillatornetze könnten somit als biologisch motiviertes neuronales Substrat verwendet werden, auf dessen Grundlage verschiedene “ Wahrnehmung als Inferenz ”-Theorien fundiert werden koennen. Sie weisen weiterhin neuartige Wege zur Lösung von Constraint Satisfaction Problemen.

Der zweite Teil der Arbeit befasst sich mit dem Bau von neuromorphen Systemen und die physische Umsetzung der neuronal inspirierten Algorithmen. Ich beginne mit der Beschreibung des ”address event representation” (AER) Protokolls für Eventkommunikation und präsentieren eine automatisierte Methode für die Synthese von AER Schnittstellenschaltungen auf Basis von Behavioural VHDL-Code. Diese automatisch synthetisiert AER Schnittstellenschaltungen sind Teil der notwendigen Kommunikationsinfrastruktur der ereignisbasierte Rechenarchitektur, die ich entwickelt habe. Weiterhin präsentiere ich eine Hybride CMOS-Memristor-Architektur, in

welcher Memristoren als plastische synaptische Elemente dienen und präsentiere messungen einer physischen Implementierung dieser Architektur.

Einige der schwersten und praktisch anwendbarsten Probleme in den Bereichen der Informatik, Optimierung und des machine learnings können als ”best match”-Probleme formuliert werden, bei denen es das Ziel ist, die Werte für eine grosse Zahl Variablen zu finden, welche ein Set aus Bedingungen maximal erfüllen. Es existiert eine fundamentale Beziehung zwischen diesen Problemen und der Relaxation zu minimalen Energiezuständen in physikalischen, stochastischen Systemen. Der wichtigste Beitrag dieser Arbeit ist es, eine neue fundamentale Verbindung zwischen ”best match”-Problemen und der Dynamik von Oszillatorsystemen zu ziehen. Ich zeige, dass Event-basierte Systeme von gekoppelten Oszillatoren in VLSI welche diese neuen Dynamiken ausnutzen ein klares Potenzial haben, bestehende von-Neumann Architekturen in Bezug auf Geschwindigkeit und Stromverbrauch beim Lösen von verschiedenen Arten von ”best match” Problemen zu übertreffen, sofern sie ein effizientes und verteiltes Event-Routing-Schema verwenden. Simulationen zeigen beispielsweise, dass die vorgeschlagene VLSI-Architektur Instanzen des breit anwendbaren boolean satisfiability Problems um drei Gröenordnungen schneller lösen kann als ein herkömmliche Computer bei vergleichbarer Durchschnittsfrequenz. Zusätzlich zu ihrem technologischen Potenzial, motivieren die bei der Lösung des ”best-match” Problems verwendeten neuartigen Dynamiken eine neue nicht-stochastischen Formulierung von ”Wahrnehmungsinferenz” Phänomenen, die diese Phänomene mit kortikalen rhythmen in Bezug setzen anstatt mit neuronalem- oder synaptischen Rauschen.

Disclaimer

I hereby declare that the work in this thesis is that of the candidate alone, except where indicated in the text, and as described below.

Chapter 2 is based on the paper "Sequential activity in asymmetrically-coupled winner-take-all circuits" by Hesham Mostafa and Giacomo Indiveri Mostafa & Indiveri (2014).

Chapter 3 is based on the paper "Recurrent Networks of coupled WTA-oscillators for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri Mostafa et al. (2013b).

Chapter 4 is based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri Mostafa et al. (2015c)

Chapter 5 is based on the paper "Automated synthesis of asynchronous event-based interfaces for neuromorphic systems" by Hesham Mostafa, Federico Corradi, Marc Osswald, and Giacomo Indiveri Mostafa et al. (2013a)

Chapter 6 is based on the paper "Implementation of a spike-based Perceptron learning rule using TiO_{2-x} memristors" by Hesham Mostafa, Ali Khiat, Alexander Serb, Christian Mayr, Giacomo Indiveri, and Themis Prodromakis Mostafa et al. (2015a)

Chapter 7 based on the paper "An event-based architecture for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Muller, and Giacomo Indiveri Mostafa et al. (2015b)

Table of Contents

1

Introduction

1.1. Introduction and Overview

The emergence of cognitive faculties such as memory, subjective experience, and intentionality from a physical entity such as the brain is a baffling phenomenon for which no adequate explanation yet exists. One way of making this phenomenon a proper subject of investigation is to assume the brain is a computing device and that cognitive faculties are a product of complex computations. This "brain as a computer" hypothesis is a widely, though not unanimously (Penrose, 1999), accepted hypothesis. This hypothesis is crucial in the field of computational neuroscience as it holds out hope that the formal set of rules, or the algorithm, that neural circuits employ to efficiently execute complex cognitive functions can gradually be discovered, and eventually be used to engineer efficient computing systems. To achieve the later goal, insights from the field of computational neuroscience alone are not enough. Knowledge from the fields of computer science and electrical engineering have to be brought to bear to translate ideas from computational neuroscience into concrete computing systems that can efficiently solve real-world problems. This multi-disciplinary undertaking is what the field of neuromorphic engineering is about.

The multi-disciplinary nature of neuromorphic engineering makes it a prime research area in which to investigate the possible technological ramifications of brain research. However, this same multi-disciplinary nature can prove problematic when workers in one discipline are unaware of the limitations and particular goals of the other disciplines. For example, biological realism and the ability to generate testable predictions about the behavior of biological networks is naturally the main goal when computational neuroscientists develop network models. Computer scientists and machine learning researchers do draw inspiration from biologically-realistic network models. However, they develop neural networks primarily based on algorithmic considerations and fast execution times on conventional computers. Electrical and neuromorphic engineers on the other hand are more interested in how biologically-inspired network models might inform the design of novel, more efficient physical computing systems. The 'physical realizability' of network models on engineered substrates that are noisy and that exhibit component-to-component mismatch and have limited wiring resources, such as very large scale integrated (VLSI) chips, is an important factor that is often not considered by computational neuroscientists or computer scientists.

Efforts for developing efficient brain-inspired physical computing architectures thus have to reconcile ideas from multiple fields, but these ideas were often developed with different sets of limitations and goals in mind, making it hard to achieve a meaningful synthesis. To achieve such a synthesis, an integrated approach is needed where experimental neuroscience results provide a starting point, and a source of inspiration, for developing neural network architectures that simultaneously meet the requirements of computational efficiency and physical realizability. The main goal of this thesis is to demonstrate the value of such an integrated approach in developing brain-inspired computing systems. The networks and hardware architectures developed in

this thesis address two main computational tasks: the implementation of large-scale symbolic computation, and searching for maximally consistent input interpretations. These two modes of computation are central in the discussion of biological computation where they are used to account for a number of cognitive phenomena (Fodor & Pylyshyn, 1993; von Helmholtz & Southall, 1925; Gregory, 1980; Friston, 2003). In cognitive science, an intense debate revolves around which of these two modes of computation is employed by the brain (Piccinini & Scarantino, 2011). One side argues that at a high level, the brain is operating on symbolic representations (the classical computationalist view (Newell & Simon, 1976; Fodor & Pylyshyn, 1993)) while the other side argues that cognition is inherently non-symbolic and the relevant mode of computation is the relaxation of a neural network to the maximally consistent (or "harmonious" (Smolensky, 1986)) interpretation of its input (a part of the sub-symbolic or connectionist view (Rumelhart & McClelland, 1986; Rumelhart, 1998; Feldman & Ballard, 1982)).

In this thesis, we are mainly interested in developing network models that can explain in mechanistic terms how the brain can carry out either form of computation, and how these network models may be used to develop efficient physical computing systems. The network architectures we describe, however, can still be relevant to this debate as they can be used as biologically motivated neural substrates on which to ground various abstract computational theories of the brain. I now give a brief overview of these two modes of computation.

Early versions of the "computational theory of mind" argue that the brain, at a high level, carries out symbolic computation, i.e, that it maintains a symbolic representation of the world and of its internal states and that it operates on this symbolic representation in a manner analogous to a digital computer, or analogous to its more idealized version, the Turing machine (Newell & Simon, 1976; Fodor, 1975; McCulloch & Pitts, 1943). As a "physical symbol system" (Newell & Simon, 1976), the brain is thus able to compute any practically computable function according to the Church-Turing thesis. While this universal computing ability is an attractive feature, there is no clear model that can explain in mechanistic terms how biologically-realistic networks may support large-scale symbolic computation. A notable difficulty is reconciling the localizable nature of symbols and the discrete-time nature of symbolic computation with the distributed, analog, and continuous-time nature of a realistic neural network.

The connectionist or sub-symbolic view of cognition discounts the symbolic representation view and argues instead that cognitive functions simply arise from the distributed activation of units in a neural network, which is not reducible to a set of symbols and symbol-manipulation rules. This view has been vigorously championed by the parallel distributed processing (PDP) group in the eighties (Rumelhart & McClelland, 1986; Rumelhart, 1998) that often used sensory perception phenomena to support their views (Smolensky, 1986). By discarding the symbolic paradigm, the PDP group could develop networks where the basic operation is the relaxation to a state or set of states that are maximally consistent with the applied input, where consistency is judged according to an internal model encoded in the network connectivity. Such a network scheme naturally explains the crucial inferential aspect of sensory perception that consists of finding the maximally consistent interpretation of noisy, incomplete, and multi-modal sensory input using a model of the world constructed from experience. While connectionist schemes can more naturally explain perceptual phenomena as compared to symbolic schemes, they do not perform well in explaining various symbolic forms of reasoning that are crucial in language processing for example (Fodor & Pylyshyn, 1993).

The first part of this thesis describes how these two models of computation, the symbolic processing model and the relaxation to maximally consistent state model, can be implemented through the dynamics of biologically-realistic neural networks: Chapter 2 describes a model for the generation of noise-robust and input-sensitive patterns of sequential activity in neural networks. This network model forms the basis for the implementation of large-scale and state-

dependent symbolic computation. Chapters 3 and 4 describe how oscillatory dynamics, which are ubiquitous in the brain, allow a neural network to represent a consistency model and execute a non-stochastic and noise-robust search for the maximally consistent interpretations of ambiguous inputs. I show in chapter 4 that such networks can be used to model various perceptual inference phenomena such as perceptual multi-stability and the explaining-away effect. In addition to biological fidelity, the networks I describe in chapter 4 were developed so that they can be efficiently implemented on mixed-signal VLSI chips. The second part of the thesis addresses the implementation of these networks on physical devices as well as well as how their performance compares to standard computer science approaches used to search for maximally consistent states.

The second part of the thesis begins by addressing some of the engineering aspects involved in developing neuromorphic VLSI systems. Chapter 5 discusses a novel approach for rapidly developing address event representation (AER) communication circuits in neuromorphic chips and chapter 6 discusses hybrid CMOS-memristor systems. Chapter 7 picks up again the main thread of this thesis. The second model of computation described in the first part of the thesis, the relaxation to maximally consistent state model, is directly related to some of the most difficult and practically useful problems in the fields of computer science and optimization. The goal in many of these problems is to find the configuration of a large number of variables that is maximally consistent, i.e, that maximally satisfies a set of constraints. These problems, which are called Constraint satisfaction problems (CSPs), have wide applicability in areas such as channel coding (MacKay, 2003), circuit optimization (Kirkpatrick et al., 1983), and scheduling (Garey et al., 1976). In chapter 7, I describe the *inferenceEngine* architecture, a hardware architecture that was developed based on the biologically realistic oscillatory network model described in part one of the thesis. I show that this biologically-inspired architecture can be used to efficiently solve a number of CSPs such as Boolean satisfiability problems and graph coloring problems, and that it can potentially outperform conventional digital processors that use state of the art algorithms. In chapter 8, I show that the *inferenceEngine* architecture can be used to accurately reproduce the dynamics of stochastic networks like Restricted Boltzmann Machines (RBMs) and that the architecture represents an efficient scheme for implementing this class of networks on custom VLSI chips.

1.2. Motivation

Digital computing devices built using standard logic gates are the standard systems for carrying out computing tasks nowadays. These systems employ a computational model which consists of the manipulation of symbols or bits stored in memory according to a set of rules with internal states, or a program. They represent a physical embodiment of the abstract concept of a Turing machine. It has been argued that these computing systems are universal, in the sense that, given arbitrarily large memory (as in a Turing machine), they can compute any physically computable function. The long theoretical tradition behind the Turing-machine model of computation, and the success of devices embodying this model, has conditioned many computer scientists and electrical engineers to an exclusively Turing-machine like view of what constitutes a computing device. This view is, however, severely limited. While no one can convincingly argue that a physical system embodying an alternative model of computation is able to solve problems that a conventional computer with sufficient memory can not, one can certainly argue that such an alternative system may very well solve particular problems much more efficiently than a conventional computing system made of logic gates and digital memory elements.

A central a-priori assumption of the work in this thesis is that by adopting a broader view of

what a computing device is, we would be able to devise problem-specific computing architectures that outperform conventional computers in particular domains. This is motivated by the observation that the natural dynamics of many physical systems constitute efficient algorithms for solving some difficult computational problems. For example, an Ising magnetic spin model naturally solves a difficult constraint satisfaction problem simply by relaxing to its ground energy state (Barahona, 1982). This has motivated the development of computing architectures that operate according to similar principles for solving hard combinatorial optimization problems (Johnson et al., 2011). Similarly, speedups that are unachievable on classical computers in integer factorization problems (Shor, 1997) and unstructured search problems (Grover, 1996) could be obtained by exploiting the dynamics of quantum systems. Many physical systems could thus be said to compute which raises the question of what then constitutes a computing device? (Copeland, 1996). The simple definition of "usable computation" expressed in ref. (Crutchfield, 1994) is the one most relevant here: a device is performing usable computation if its dynamics can be mapped to a solution algorithm for a problem we are interested in.

One of the goals of this thesis is to find a dynamical system that is implementable on VLSI devices whose behavior constitutes a solution algorithm to a computational problem of relevance. These dynamics should be more efficient in terms of speed and power consumption than comparable algorithms running on standard digital computers. To achieve that, we look to biological networks for inspiration. This is motivated by the fact that evolutionary optimization has rendered such networks extremely efficient in solving a number of computational problems that are relevant to survival. It is also safe to assume that such networks do not operate in the same way as a conventional computer, so we will be able to draw from them novel forms of usable computation that can not be directly or trivially reduced to digital computation. The later assumption is supported by the overwhelming performance advantage animal brains have in analyzing and making decisions in unstructured environments compared to conventional computers. This suggests that the two use qualitatively different computing approaches. Now that we have chosen to focus on the dynamics of biologically realistic networks in the search for efficient forms of usable computation, we have to consider problems where such networks might have a performance advantage compared to conventional computers.

One particular computational problem at which biological networks excel is finding correct interpretations of incomplete and ambiguous sensory inputs. A long theoretical tradition casts sensory processing as a process of inferring the maximally consistent interpretations of the sensory inputs according to an internal model of the environment which is constructed based on prior experience (von Helmholtz & Southall, 1925; Gregory, 1980; Friston, 2003). This process can be formulated as solving a constraint satisfaction problem or a probabilistic inference problem. These problems are highly relevant in computer science and machine learning. A large part of this thesis thus focuses on how these problems can be solved by biologically realistic recurrent networks and by custom VLSI chips that were developed to capture the relevant dynamics of these networks.

As well as enabling the development of new efficient computing architectures, investigating the forms of usable computation that can be obtained from the dynamics of recurrent networks can also shed some light on the nature of computation in biological networks. Another goal of this thesis is thus to develop hypotheses and testable predictions that could be used to elucidate how biological networks compute. Developing such hypotheses is crucial for driving advances in computational neuroscience. The pure bottom-up approach where results from experimental neuroscience alone inform the investigation of biological computation is hampered by the complexity of the brain that makes it difficult to draw unifying insights from the data, as well as observability limits that preclude detailed large-scale measurements that could expose unifying

patterns. As in other natural science domains such as particle physics, top-down hypotheses are needed to enable the design of hypothesis-directed experiments and to provide a framework in which to analyze and interpret experimental data. In this thesis, I will develop hypotheses regarding the neural substrates of two forms of biologically relevant computation: the search for consistent interpretations of ambiguous inputs, and large-scale symbolic computation.

1.3. Features of Neurally Inspired Computation

A biological neuron is a complex dynamical system that integrates incoming input in a highly non-linear manner. Even though each neuron has a complex dynamical state, neurons communicate using all-or-nothing stereotypical spikes, where each spike carries relatively little information about the state of the neuron that emitted it. The effect of a spike on a downstream neuron is far from repeatable in many cases due to synaptic transmission failures and fluctuations in the number of released synaptic vesicles. Neurons themselves are highly heterogeneous and exhibit markedly different stimulus responses and connectivity profiles. The predominantly local connectivity structure in biological networks mean that each neuron receives spikes from a small subset of other neurons and thus has no direct access to the activity of most of the neurons in the network.

These features of biological networks would be perceived as limitations and non-idealities if these networks were executing conventional algorithms that depend on bit-perfect computations and global accessibility of data. That is why it is important to discard pre-conceived notions of algorithms as sequences of instructions and to think of biological algorithms as they relate to the actual dynamics of recurrent neural networks, not as they relate to any abstract model of computation. This view is in line with the approach advocated by Hopfield: "our understanding of biological computation and its origins must come through studying the relation between computation and its underlying hardware, not computation as a logical structure" (Hopfield, 1994).

If we thus think in terms of the biological neural hardware, we would require the forms of usable computation that we derive from the dynamics of recurrent neural networks to be tolerant of intermittent communication failures between the network elements. These forms of computation should also accommodate heterogeneity and parameter mismatch among the network elements. Ideally, we would want these forms of usable computation to actively exploit these seemingly limiting features, instead of simply accommodating or working around them. We will illustrate this when discussing neurally inspired schemes for solving constraint satisfaction and inference problems where communication failures and parameter mismatch among the neurons have beneficial effects.

Perhaps the most crucial feature of biologically-realistic neural networks, and one that should be exploited by any form of usable computation that we draw out of them, is their massively parallel and distributed operation. Each neuron or neuron-like element represents a simple processing unit that processes its input spikes and generates output spikes independently of all others neurons. One consequence of such de-centralized operation is the absence of single points of failure. If we devise a form of usable computation that embraces this massive parallelism, we would expect its performance to degrade gracefully when network elements malfunction as I will illustrate when discussing neural algorithms for solving constraint satisfaction problems.

Surprisingly, many problems whose solution must satisfy some global requirements and many algorithms that presume the availability of global data signals or global synchronization signals can be recast as networks of independent elements with strictly local connectivity. Two examples that we will encounter in this thesis are neural local search algorithms that find globally

optimal solutions to constraint satisfaction problems, and self-timed symbolic networks that are able to achieve global synchronization of the flow of symbols. Neurally inspired physical computing systems are thus in a position to reap the advantages of highly parallel operation and strictly local communication requirements, while still being general enough to solve many classes of relevant problems.

1.4. Von Neumann Architectures and Neurally Inspired Computation

The von Neumann architecture is a practical realization of the abstract Turing machine model of computation. It is characterized by separate execution and memory units. The execution unit reads and stores data in the memory unit. The behavior of the execution unit is governed by a set of instructions (the program) which the execution unit also reads from memory. These instructions can modify local registers in the execution unit, read and modify locations in memory, or acquire inputs or generate outputs from/to peripheral devices. The von Neumann architecture forms the basis of almost all modern computers. Von Neumann architectures differ in many ways from biological networks and biologically-inspired computing systems.

Perhaps the most fundamental difference is in the programming model used. The knowledge, or the program, in a neural network is encoded in the network architecture, and in the weights of the connections between the neurons. This programming model is not as transparent as the sequence of instructions model employed by von Neumann architectures; it is typically harder to design a neural network to exhibit a certain behavior or to reason formally about such behavior compared to writing a formally correct program for a von Neumann computer. This disadvantage, however, is in many cases outweighed by the ability of neural networks to learn or "self-program" based on input data and error signals. Since the network program is stored in the weights, changing the weights changes the program. By changing the weights based on the activity of the neurons in the network, and since this activity is input-dependent, the weight changes will reflect some properties of the network input. This can be used to learn a probabilistic model of the input data (Ackley et al., 1985; Nessler et al., 2013), uncover structure in complex time series (Jaeger, 2002), or to learn to classify input patterns (Krizhevsky et al., 2012).

While modern computers based on the von Neumann architecture typically employ multiple processing cores, the level of parallelism achieved by biological networks is still far beyond that achieved by conventional computers. One crucial reason behind the higher level of parallelism in biological networks is that computational and memory primitives in biological networks are co-localized. The processing elements, the neurons, and the memory storing elements, the synapses, are tightly coupled. The synapse is not merely a memory element as its weight can change due to the activity of its source and target neurons making it a processing element in its own right. Memory need not only be stored in synaptic weights, it can also be stored short-term in the persistent activity patterns of the neurons in the network. These activity patterns can be maintained through recurrent excitation. It is thus difficult to clearly distinguish between memory elements and processing elements. This is indicative of the tight coupling between the two.

Neural networks thus avoid one of the major performance-limiting features of von Neumann architectures: the need to shuttle data and instructions back and forth between a processing element and a central memory. This continuous shuttling of information creates the so-called 'von-Neumann bottleneck' where the execution speed of the von-Neumann architecture is not

limited by how fast the processing elements can execute instructions, but rather by how fast the processing elements can get data and instructions from memory. Several approaches exists to alleviate the effects of the von-Neumann bottleneck such as the use of fast cache memories that maintain a copy of some of the data and instructions in main memory, thereby reducing the number of times that the processing elements need to access the larger, but relatively slower main memory. While extensive effort has gone into optimizing cache hierarchies and cache prefetching and replacement algorithms, none of these efforts can completely get rid of the problem in the same radical way as biological networks do.

The co-localization of computation and memory primitives alone is not enough to enable massively parallel operation. Information needs to flow between the processing elements to realize any meaningful computation. A communication scheme is thus needed that will not bottleneck the parallel operation of the network. The processing elements, the neurons, are however extremely numerous in biological networks which would place very difficult speed demands on any general communication scheme. In neurally-inspired computing architecture as well, the processing elements will typically be simpler and more numerous than the processing cores used in conventional computers. To enable large numbers of processing elements to efficiently communicate, each element should communicate mainly with its local neighborhood . This is indeed the case in biological networks where connectivity patterns are predominantly local. This is reflected in the functional organization of the brain where the functional specialization of an area puts neurons that collectively serve a particular function closer together, reducing the need for long-range communication. Only when communication is predominantly local can we reap the advantages of utilizing a large number of parallel processing elements as information can flow in parallel within a large number of local domains. Predominantly local communication is more energetically efficient. It also reduces wiring resources in biological networks which translates to smaller address spaces in engineered neural systems as addresses can be reused in different local domains. While the efficiency of a program running on von-Neumann architectures can be improved by keeping related data and instructions nearby in memory, locality constraints are not enforced in the programming model. Consequently, von-Neumann architectures can not reap the energy and speed advantages of parallel local communication that neural architectures enjoy.

1.5. The Promise of Neuromorphic Engineering

Even though the human brain was always a source of inspiration for building computing devices, the formal theory of computation (Sipser, 1996), and the physical computing systems developed according to this theory, have traditionally had very little to do with the principles of operation of the brain. Lately, however, there is a growing interest in developing neuromorphic systems that explicitly attempt to model various aspects of how the brain works. The architectures of these systems reflect the general organizational principles of nervous systems. These systems are organized as populations of excitatory and inhibitory spiking neurons with configurable synaptic connections. There is a diverse variety of such systems being developed (Navaridas et al., 2013; Merolla et al., 2014b; Qiao et al., 2015; FACETS, 2005–2009; Benjamin et al., 2014; Park et al., 2014), some of them with industrial backing (Merolla et al., 2014b), which differ based on whether they operate in the digital or the mixed-signal (analog/digital) domain, whether they run at biological or accelerated time scales, and whether they employ plastic or fixed-weight synapses.

The superior performance and energy efficiency of the brain are often touted as reasons for building such neuromorphic systems and for why these systems can eventually be viable alterna-

tives to conventional computers in some problem domains. The brain is an incredibly complex system, however, with complex molecular machinery and complex dynamics operating at many different time scales. Neuromorphic systems capture in a very coarse and general fashion how biological networks operate. The claim that at this coarse level of detail, neuromorphic systems can reproduce, even partially, the immense efficiency advantage that biological computation exhibits in some problems is highly doubtful given the hugely deficient biological fidelity of these neuromorphic systems. The claim is made more doubtful by the fact that such neuromorphic systems rest on no clear computational principles from either the field of computer science and machine learning or the field of computational neuroscience.

Even if we assume that at some point in the future, we can fully elucidate the mechanisms the brain employs to execute particular cognitive functions, it remains doubtful if such mechanisms could be efficiently mapped directly to neuromorphic VLSI devices. That is because the physical substrate employed by the brain is vastly different from silicon. For example, each individual neuron is a complex spatially extended structure with morphology-dependent dynamics. It is doubtful whether such a distributed structure could be efficiently and faithfully modelled using transistor circuits. If it is unlikely that we will stumble upon brain-level performance by using neuromorphic systems that implement hugely simplified neuron and synapse models, and if it is also unlikely that we will be able to directly implement a computational model formulated in terms of realistic neuron and synapse dynamics efficiently on silicon devices, then the question arises of how to approach the problem of building brain-inspired computing architectures that are viable alternatives to conventional computers.

One approach that I follow in this thesis is to first identify the relevant collective dynamics of biological networks that might allow them to efficiently solve a particularly challenging computational problem such as constraint satisfaction and finding consistent interpretations of ambiguous inputs. I then develop biologically-realistic network models to investigate these dynamics and to generate testable predictions that could be used to verify whether these dynamics are actually employed by the brain to solve the problem under investigation. The final step involves simplifying these biologically-realistic dynamics as much as possible to allow efficient implementation on custom silicon chips. This simplification step almost completely destroys the biological fidelity of the network model. However, it still keeps the attractive features of neurally inspired computation that I described in the previous subsections as well as the general spirit of the biologically realistic model. The efficiency of the hardware-friendly dynamics that come out of the simplification step are compared to conventional algorithms running on digital computers and changes are made to the dynamics to make them competitive with conventional algorithms. This comparison step also generates clear requirements on how fast a hardware implementation needs to run to outperform a conventional processor. The end result is a biologically realistic model that can shed some light on how biological networks compute, as well as a hardware architecture that is inspired by this model but whose dynamics have been optimized for the silicon substrate and tuned so as to compare favorably with conventional algorithms.

Part I.

Usable Computation from Neural Dynamics

2

Sequential Activity in Neural Networks

This chapter, except for section 2.5, is based on the paper "Sequential activity in asymmetrically-coupled winner-take-all circuits" by Hesham Mostafa and Giacomo Indiveri, Neural Computation, 2014.

One of the hallmarks of cortical processing is the sequential processing of sensory stimuli in multiple cortical areas in order to realize complex transformations of the input. One example is the visual processing stream where visual input arriving from the LGN is successively processed in multiple stages involving V1, V2, and higher areas such as IT (Van Essen et al., 1992). The decomposition of complex transformations into a cascade of simpler ones allows the reuse of the outputs of lower processing stages in order to more efficiently realize multiple subsequent transformations. For example, the visual input transformations carried out in areas V1 and V2 are reused in multiple distinct processing streams, the ventral stream and the dorsal stream (Goodale & Milner, 1992). The implementation of these two processing streams is rendered more efficient by sharing the initial common processing stages: V1 and V2, thereby avoiding duplication of resources, and allowing top-down modulations of the early visual areas to be reflected in multiple processing streams (Markov & Kennedy, 2013).

The decomposition of complex transformations also allows the outputs of multiple lower level processing stages to converge onto one high-level processing stage. For example, several brain areas display surpamodal, or modality-insensitive, representations of perceived concepts or actions (Ramsey et al., 2013; Fairhall & Caramazza, 2013). These could be achieved only if outputs of multiple sensory areas feed into the supramodal area. Again this avoids duplication of resources by first transforming modality-specific information into a modality-invariant form for subsequent processing.

Cascaded processing of information or bits is at the heart of modern digital computers. Modern processors for example typically employ pipelined processing schemes where the execution of each instruction is broken down into multiple stages. Such a pipelined scheme allows higher instruction throughput as multiple instructions can be simultaneously present at different stages in the pipeline, removing the need to completely process one instruction before accepting the next. Like the brain, digital processors also employ convergent and and divergent processing streams. A central challenge in designing such processing streams in conventional processors is making sure that a processing stage only communicates completely processed data (and not some intermediate erroneous state) to the next stage. Global clocks are thus used to make sure successive stages process data in a well-defined pipelined manner. Such a global clock mechanism is biologically unrealistic. This raises the question of how neural networks with local connectivity can support reliable cascaded information processing where inputs arrive from

multiple sources at arbitrary times, and where each neural processing stage exhibits variable and input-dependent delays.

In this chapter, I investigate a local neural mechanism that can support the reliable processing of multiple streams of information through successive stages. At each stage, the input activity pattern could be transformed in various ways, and only when the transformation is complete, is the next stage activated to process the results of the previous stage. If a stage is "busy" carrying out a transformation, it does not process new inputs and the previous stages halt, i.e, stabilize at attractor states, waiting for the busy stage to become "ready". A stage receiving input from multiple sources waits until all inputs are available before carrying out the transformation. I will describe how this "neural flow control" mechanism can be used to realize well-defined state-dependent symbolic computation.

The networks implementing this neural flow control mechanism were inspired by the asymmetric inter-layer connectivity profile observed in cat and rat neocortices (Binzegger et al., 2004; Thomson et al., 2002a; Watts & Thomson, 2005). This chapter begins by analyzing the general dynamics of networks exhibiting this asymmetric connectivity profile. I show that these networks can be used to realize flexible and noise-robust patterns of sequential activity. I extensively analyze these sequential activity dynamics, both in rate-based and in biologically-realistic spiking network before using them to implement networks that can carry out large-scale state-dependent symbolic computation. I begin with a general overview of various schemes for realizing sequential activity patterns or sequential transformations of inputs.

Computational models of networks that exhibit sequential patterns of activity typically make use of asymmetric connections that guide activity from one neuron pool to the next. For example, asymmetric feed-forward excitatory connections are a prominent feature in synfire chain models (Abeles et al., 1982), while asymmetric inhibitory connections have been shown to play an important role in meta-stable networks of competing neurons used to generate patterns of sequential activity (Rabinovich et al., 2001; Afraimovich et al., 2004; Rabinovich et al., 2008). In unstructured networks that do not have explicit asymmetric connectivity patterns, short or long term plasticity mechanisms can lead to the formation of asymmetric connections which can then give rise to sequential patterns of activity (e.g., see (Fiete et al., 2010)). Competitive recurrent network models can use different types of adaptation mechanisms to produce sequential patterns of activity (Hopfield, 2010; Deco & Rolls, 2005; Verduzco-Flores et al., 2012). The adaptation mechanisms present in these networks reduce the activity of the active group of neurons in the network so that a different set of (non adapted) neurons can win the competition and suppress the adapted set of neurons (which then can slowly recover from the adaptation).

To generate patterns of sequential activity that can be halted and resumed by external input, a network requires a structure of stable fixed points of activity, or *attractors*, which can store the current state of the sequence. Sequential activity in attractor networks like the Hopfield network (Hopfield, 1982) can be obtained by introducing asymmetric connections that guide the activity from one attractor to the next (Amit, 1988; Kleinfeld & Sompolinsky, 1988). During spontaneous sequential activity, the asymmetric connections should be slow, or exhibit a delay that is long enough to allow the fast symmetric connections to stabilize one attractor before the asymmetric connections induce a transition to the next attractor. If the asymmetric connections are weak enough, then the transition between attractors can be triggered and controlled by the application of external input signals. The Hopfield network dynamics, however, are highly simplified and it is questionable whether the phenomena arising from these dynamics can be reproduced in biologically realistic network models.

The network model I present in this chapter makes use of asymmetric connections to induce transitions between attractors, but it does not require the artificial separation between slow and fast synapses, or the fine tuning of synaptic weights. The network I propose has a cortically

inspired architecture with realistic neuronal and synaptic dynamics; it exhibits stable transitions from one state to another; and it can learn to reproduce temporal sequences imposed by the input signals by making use of plastic feed-forward excitatory synapses.

2.1. The Winner-take-all Circuit

The basic building block of the proposed network is the Winner-Take-All (WTA) circuit shown in Figure 2.1. WTA circuits can have multiple attractor states, characterized by heightened activity in a single excitatory population that persists after external input is removed. The specific attractor state chosen is a function of the inputs received by the different excitatory populations. The network imposes an interpretation on the noisy or incomplete input by displaying one of a number of stereotypical responses. The connectivity patterns of these WTA networks are consistent with both intra-cellular recordings in cat visual cortex (Douglas et al., 1989; Douglas & Martin, 1992) and anatomical connectivity data measured in cat visual cortex and rat auditory cortex (da Costa & Martin, 2010; Martin, 2011) which point to a high level of recurrency with many excitatory, inhibitory, and excitatory-inhibitory loops (Binzegger et al., 2004; Thomson et al., 2002b). These excitatory-inhibitory recurrent loops can produce oscillatory patterns of activity if the time constant of the inhibitory population in the WTA network is made sufficiently long compared to the time constant of the excitatory populations (Ermentrout, 1992). In the presence of asymmetric connections between the excitatory populations within the WTA network, these oscillatory dynamics can facilitate the transition of activity from one excitatory population to the next by restarting the winner selection process at the peak of the inhibition. However, the requirement for slow inhibitory neurons is at odds with the observed high excitability of GABAergic inter-neurons (McCormick et al., 1985) of which many classes make synapses on the soma, axon, or proximal dendrites of pyramidal cells (Freund et al., 1983; Markram et al., 2004) leading to fast and strong inhibition. It has been argued that the WTA networks of the type depicted in Figure 2.1 represent a potential cortical circuit motif, present across the cortices of many species (Douglas & Martin, 2004).

The dynamics of different variants of WTA circuits have been extensively analyzed using rate based models (Rutishauser et al., 2012). Different WTA coupling schemes have also been proposed that can elicit spontaneous or triggered transitions between different attractor states (Rutishauser et al., 2011; Rutishauser & Douglas, 2009). This chapter extends these studies by considering both rate-based and spiking models of asymmetrically coupled WTA circuits that exhibit realistic dynamics. I introduce a novel and anatomically justified scheme for asymmetrically coupling the WTA circuits. I also investigate the role of learning and show how plastic synapses enable the network to learn to reproduce input-imposed sequences.

2.2. Asymmetric Coupling Induces Sequential Activity

The recurrent networks proposed in this chapter are composed of multiple WTA circuits, connected to each other by a coupling scheme that is based on the inter-layer asymmetric connections observed in cat and rat neocortices (Watts & Thomson, 2005). The parameters for each WTA circuit are chosen to ensure the existence of stable attractor states in the WTA circuit when it is uncoupled from the rest of the network. These attractor states provide a mechanism that restores the network activity to a well defined level at each step of the sequence. This *signal restoration* mechanism is crucial to prevent small amounts of noise in the system from extinguishing the sequential activity. Anatomically realistic asymmetric connections between the

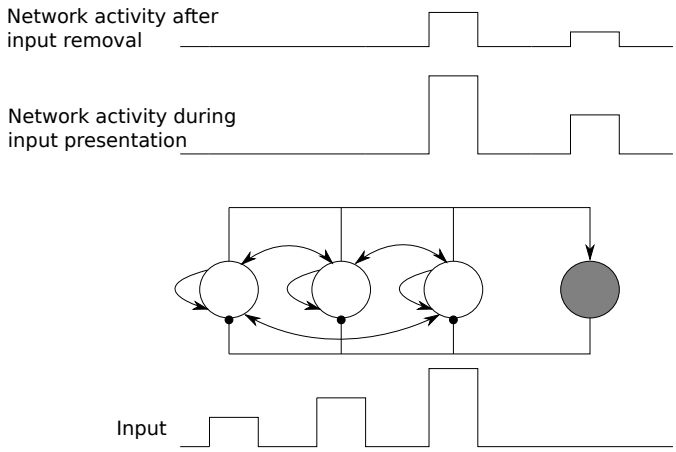


Figure 2.1: Schematic diagram of a winner take all network. Excitatory populations are shown as white circles, inhibitory populations as gray circles. The network selectively amplifies the strongest external input, suppressing the weaker ones. After the external input is removed, the network settles into the attractor state which is characterized by persistent activity in the excitatory population that received the strongest input.

WTA circuits elicit transitions between the different WTA attractor states. These asymmetrically coupled WTA circuits are thus able to sequentially visit a number of well defined states without settling into one of them and bringing the sequence to a halt. The sequence of activity produced by the network is robust to noise with little risk of the activity dying out or being driven into ill-defined states by small perturbations. The network supports the propagation of sequential activity along a number of possible paths. The path actually chosen depends on the relative strengths of the asymmetric coupling connections. If these connections are weakened, then transitions between attractors are no longer spontaneous but depend on external input to trigger them. External input can also influence the path the sequential activity takes.

Figure 2.2 shows the full connectivity profile of the proposed network. The diagram shows a network composed of three coupled stages. Feed-forward excitatory connections connect the excitatory populations of one stage to the excitatory populations of the next stage in an all-to-all fashion. The excitatory populations of one stage project back to the inhibitory population of the previous stage. There is no limitation on the number of stages that can be connected sequentially in this manner.

There is evidence that many cortical inter-layer connections are selective and asymmetric, i.e., non-reciprocated (Thomson et al., 2002a). Reconstructions of neural circuitry in the cat visual cortex point to a major loop made of non-reciprocated pyramidal-pyramidal connections (Binzegger et al., 2004). The glutamergic connections forming the loop proceed from layer 4 to layer 2/3 to layer 5 to layer 6 and back to layer 4. While the feed-forward projections from layer 4 to layer 2/3 and from layer 2/3 to layer 5 target pyramidal cells and to a lesser degree inter-neurons, there exist feedback projections from layer 5 to layer 3 and from layer 3 to layer 4 that mainly target inter-neurons (Watts & Thomson, 2005). This is consistent with the connectivity profile of our network, which is characterized by non-reciprocated feed-forward excitatory-excitatory and feedback excitatory-inhibitory connections. This assumes that the individual WTA networks are largely localized within individual layers, an assumption that is partly justified by dense intra-layer recurrent excitatory connections and the tendency of inhibitory inter-neurons to arborize locally within a single layer (Douglas & Martin, 2004).

The behavior of the network in Figure 2.2 can be qualitatively understood by considering a

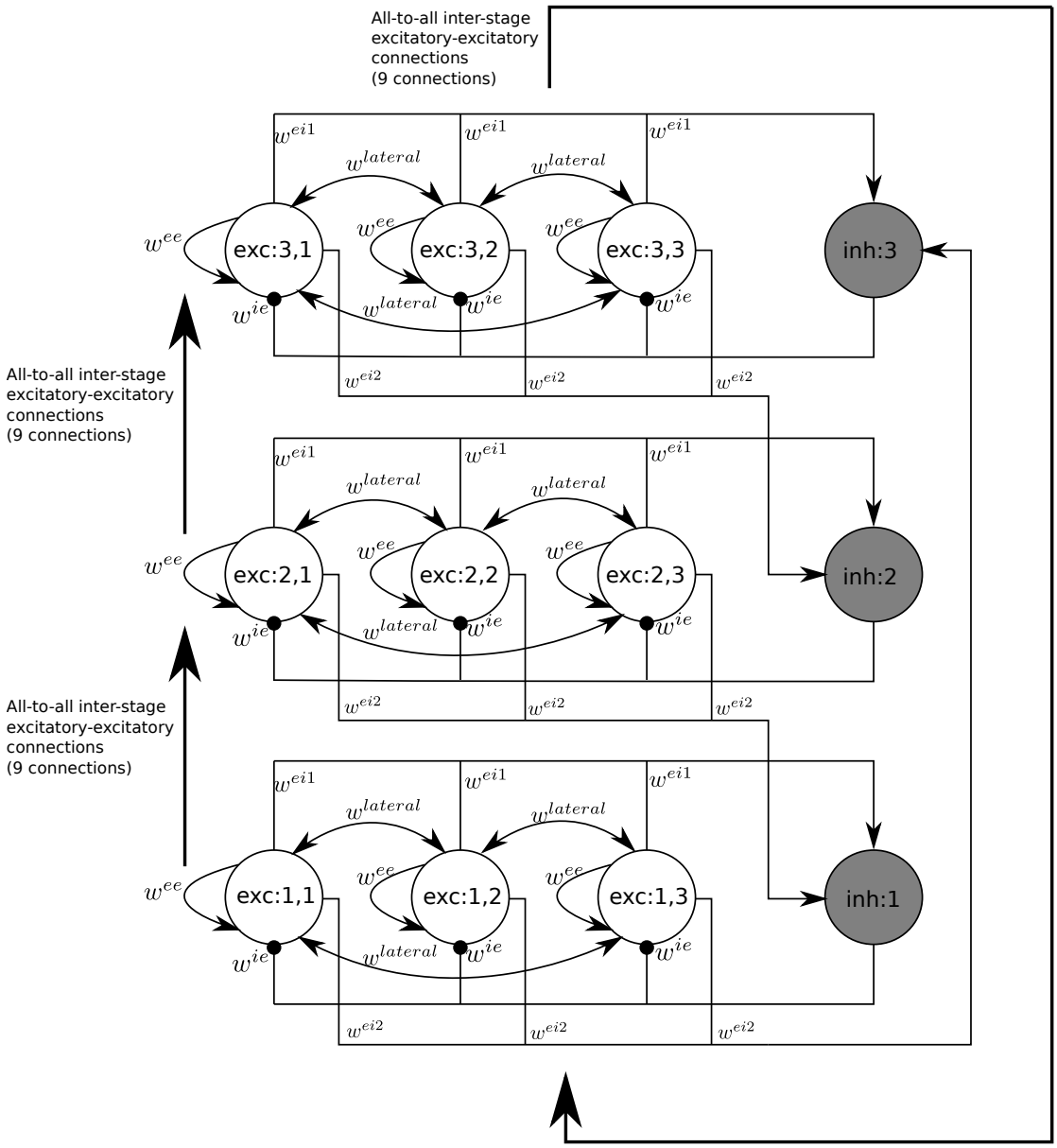


Figure 2.2: Three asymmetrically coupled WTA stages connected in a loop. Excitatory populations are shown as white circles, inhibitory populations as gray circles

single WTA stage in isolation. The excitatory populations in a WTA stage are driven by an external input and/or recurrent excitation. If some or all of the neurons in one of the excitatory populations are receiving a total input current that is larger than the current dissipated by various leak mechanisms, then these neurons will become active. Due to the recurrent excitatory connections, the firing of a subset of neurons in an excitatory population will provide additional current to other neurons in the population which will cause some of these neurons to increase their activity. Eventually the average firing rate in the population will reach a point where the synaptic current coming through the recurrent connections is large enough to maintain a self-sustaining rate which persists even after all external inputs are removed. Beyond the self-sustaining rate, the average firing rate in the population will ramp up quickly until it is large enough to trigger activity in the inhibitory population through the relatively weak excitatory to inhibitory synapses. Activity in the inhibitory population will begin curbing the runaway excitation in the winning excitatory population. Activity in the winning excitatory population will keep on ramping up, but at a slower rate. Activity in the inhibitory population will ramp up as well to match the increasing excitation. The increasing level of inhibition will shut down excitatory populations whose average firing rate is lower than that of the winning population. In an isolated, uncoupled WTA stage, the recurrent excitation in the winning population will eventually reach a stable equilibrium with the inhibition and the excitatory population will settle into a steady state firing rate. At this condition, the network has settled into a stable attractor.

If the WTA circuit is asymmetrically coupled to other WTA circuits, as in Figure 2.2, and if the connection weights are appropriately chosen, then the WTA stages will not be able to settle into stable attractors: the WTA will no longer have a constant non-zero persistent pattern of activity in the absence of external input. For example, as soon as activity in the bottom WTA circuit of Figure 2.2 becomes sufficiently high, the middle WTA circuit is activated and through the feedback excitatory-inhibitory connections, it raises the level of inhibition in the bottom WTA circuit, effectively killing the activity there.

To quantitatively analyze the network shown in Figure 2.2, I abstract the neural activity in each population and represent it by a single dynamical variable. This dynamical variable represents the average firing rate of the neurons in the population. Let R be the number of stages or WTA circuits, and C the number of excitatory populations in each WTA circuit ($R = 3$ and $C = 3$ in Figure 2.2), and let the R stages be connected in a loop as in Figure 2.2, then the full firing rate model is given by:

$$\tau_e \dot{x}^{exc}_{j,k}(t) + x^{exc}_{j,k}(t) = [w^{ee} x^{exc}_{j,k}(t) - w^{ie} x^{inh}_{j}(t) + \sum_{\substack{l=1 \\ l \neq k}}^{C} w^{lateral} x^{exc}_{j,l}(t) + \sum_{l=1}^{C} w^{f}_{j-1,l,k} x^{exc}_{j-1,l}(t) - T_e + I^{exc}_{j,k}]^{+} \quad j = 1,..R \quad k = 1,..,C \quad (2.1a)$$

$$\tau_i \dot{x}^{inh}_{j}(t) + x^{inh}_{j}(t) = [\sum_{l=1}^{C} w^{ei1} x^{exc}_{j,l}(t) + \sum_{l=1}^{C} w^{ei2} x^{exc}_{j+1,l}(t) - T_i]^{+} \quad j = 1,..,R \quad (2.1b)$$

where τ_e and τ_i represent the excitatory and inhibitory neuron time constants, $[x]^{+} = \max(0, \mathrm{x})$, and all the additions and subtractions in the subscripts wrap around to stay in the range $[1, R]$. The activity in the k^{th} excitatory population in stage j is $x^{exc}_{j,k}(t)$ and the external input to that population is $I^{exc}_{j,k}$. x^{inh}_{j} is the activity of the inhibitory population in stage j. $w^{ie}, w^{ei1}, w^{ei2}, w^{ee}, w^{lateral}$

are fixed weights (shown schematically in Figure 2.2). $w^f_{m,n,p}$ is the fixed feed-forward connection weight connecting population n in stage m to population p in stage $m+1$ where the last addition wraps around in the range $[1, R]$. If the total input to a population is clamped at x^{fixed}, then the steady state activity of this population is a linear threshold function $[x^{fixed} - T]^+$ of x^{fixed} where T, the threshold, is a population parameter. A linear threshold activation function is a good approximation of the steady state average firing rate in a population of constant leak integrate and fire neurons receiving noisy, uncorrelated inputs (Fusi & Mattia, 1999). In principle, for a step increase in mean input, the actual average firing rate in a population of neurons settles into a steady state after a number of transient modes have died out (Mattia & Del Giudice, 2002; Knight, 2000), but in Eq. 2.1 I assume the firing rate approaches steady state only through first order dynamics.

Note that the linear threshold activation function is non-saturating so any non-zero stable steady state that the network might have must be due to the interaction between the excitatory and inhibitory populations. Appendix A contains an approximate analysis of the network shown in Figure 2.2 and described by the model in Eq. 2.1.

2.3. Steering and Learning of Sequential Activity

Fig. 2.3 shows numerical simulation results when a sequence is launched by exciting the bottom WTA stage. The simulation uses the Euler method with a time step of 0.1 ms. The weights of the feed-forward excitatory-excitatory connections were independently sampled from a uniform distribution. The population parameters and connection weights are given in Table A.1 of Appendix A. The asymmetric connections destroy the fixed point attractors in each WTA stage but do not completely distort the phase space around these destroyed attractors. The regions around the destroyed attractors still exert a pull on nearby trajectories and the trajectory speed in these regions is small. These are remnants of the dynamical features of the destroyed attractor. We call these regions the "ghosts" of the destroyed attractors.

Even though there can be no sustained activity state in which one of the WTA stages is individually active, the "ghosts" of the destroyed attractors in each stage still restore the activity towards a well-defined level. The path taken by the sequential activity is a function of the feed-forward excitatory-excitatory connections linking the consecutive stages. When an excitatory population has almost "won" in stage j and is approaching the ghost of the destroyed attractor, its heightened activity is sufficient to activate the excitatory populations in stage $j+1$. Since only one winner can emerge, only the population in stage $j+1$ receiving the strongest projection weight from the active population in stage j will approach the ghost of the destroyed attractor and win the competition, and in the process kill the activity in stage j through the feedback excitatory-inhibitory connection. In other words, Activity will jump from population exc:j,k to population exc:j+1,p only if

$$w^f_{j,k,p} > w^f_{j,k,h} \quad \forall h \in \{1, \ldots, C\} - \{p\} \tag{2.2}$$

Since we connect the last (top) stage to the first (bottom) stage, the sequential activity will persist indefinitely. However, if we break the connection between the last and the first stage, the stable attractor states of the last WTA stage are restored, so that when activated, the last stage shuts down the previous stage and settles into its stable state of persistent activity. It is also possible to choose the population parameters and the intra-stage connection weights in the last stage so that activity ramps up in the last stage to a level that is sufficient to shut down the previous stage, and once the previous stage is shut down, activity decays back to zero without activating subsequent stages.

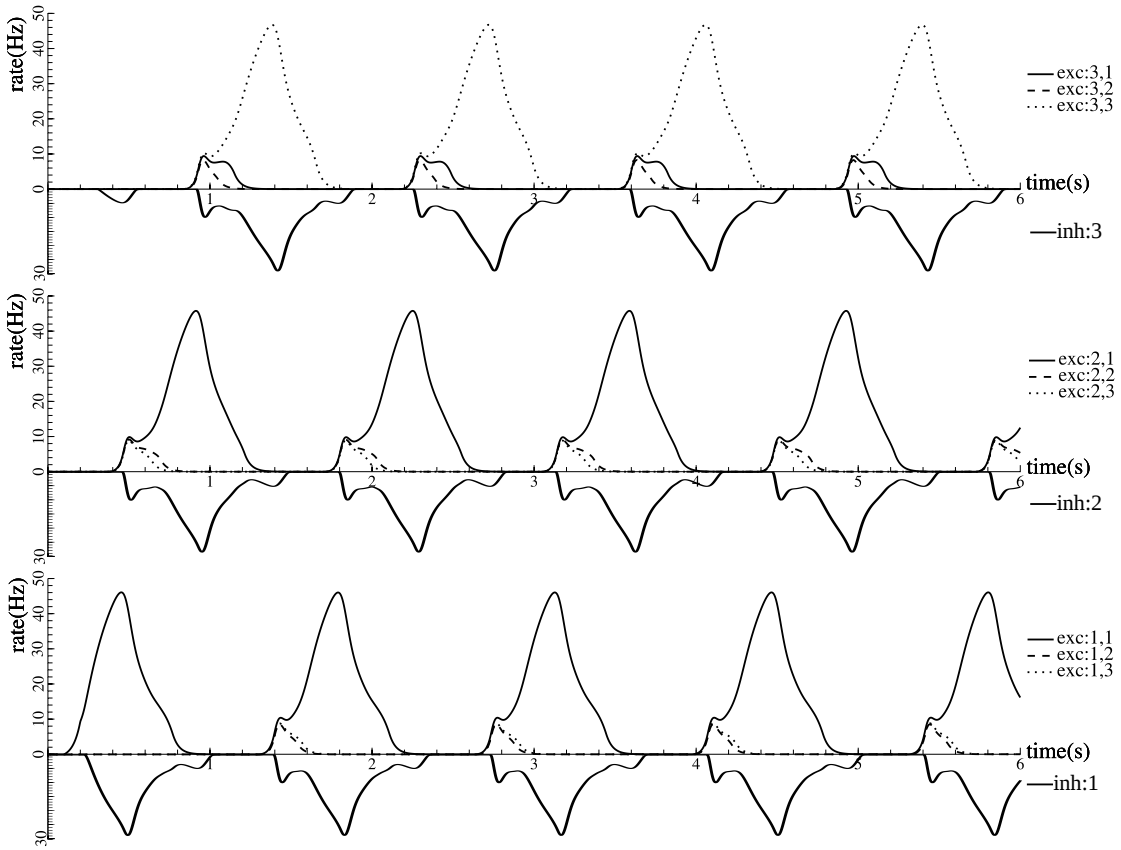


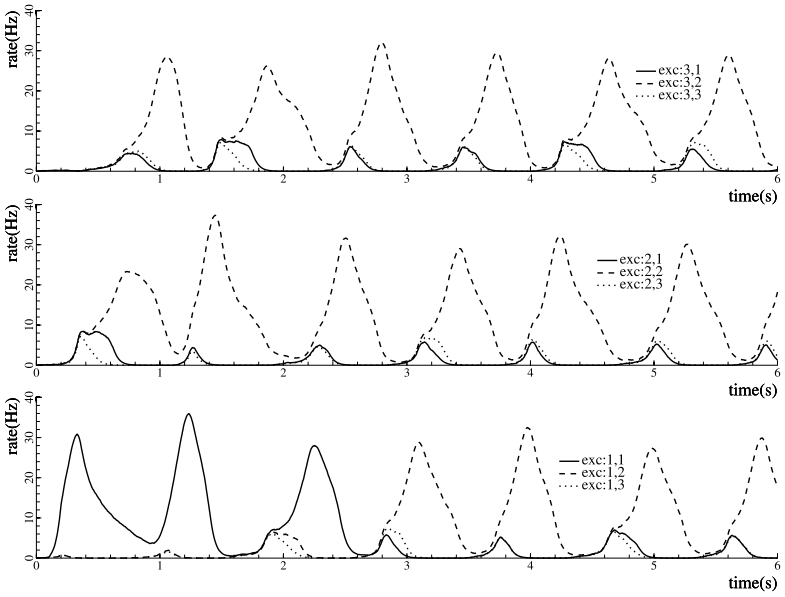
Figure 2.3: Simulation results showing the activity in each population of the network of Fig. 2.2. The activities of inhibitory populations are plotted on reversed Y-axes. A brief excitatory input is delivered to the exc:1,1 population at 0.1s to launch the sequence.

Noise Sensitivity

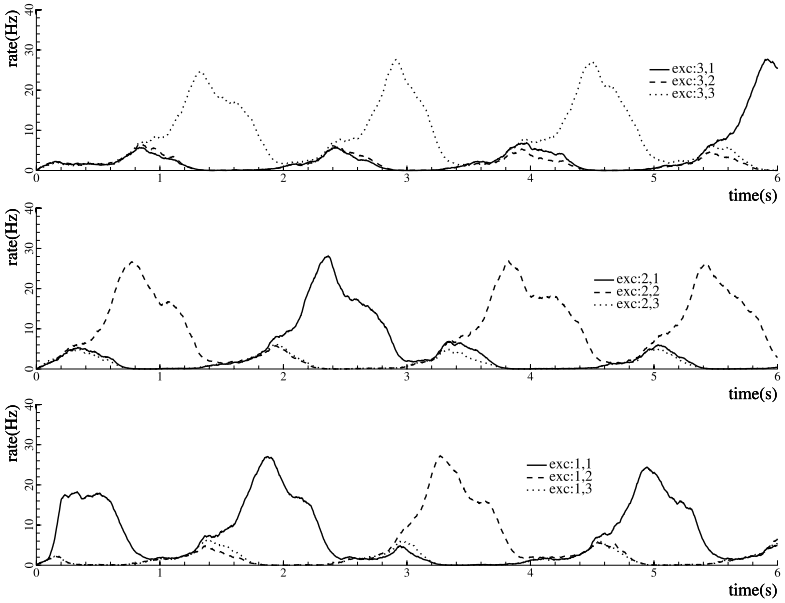
The active restoration of sequential activity at each step adds significant robustness to noise. Each step of the sequence is well defined. Activity can not propagate along two paths at once because each WTA can only support one winner, and the winning population is always unambiguous. Furthermore only by producing an unambiguous winner can a WTA stage activate the next stage and allow the sequence to proceed. Figure 2.4 shows simulation results of the network shown in Figure 2.2 at different noise levels. The populations were perturbed by uncorrelated white Gaussian noise. Integration was done using the Euler-Maruyama method with a time step of 0.1 ms. When activity is ramping up in a WTA stage, noise can bias the competition and allow an arbitrary population to win. The amount of noise needed to override the pattern of sequential activity encoded in the feed-forward excitatory-excitatory weights increases substantially if these weights are well separated. In Figure 2.4a, the noise level is not high enough to perturb the sequence. In Figure 2.4b, high noise level perturbs the path of the sequential activity but is unable to extinguish the activity, and winner selection in each stage is still unambiguous.

Triggered Transitions

If we sufficiently reduce the strength of the feed-forward inter-stage excitatory-excitatory connections, we are able to restore the attractor states of the individual WTA stages. When an



(a)



(b)

Figure 2.4: Activity in the excitatory populations of the network of Figure 2.2 when all population (including the inhibitory populations) are perturbed by a white Gaussian noise process. (a) Noise r.m.s is 300, (b) Noise r.m.s is 600

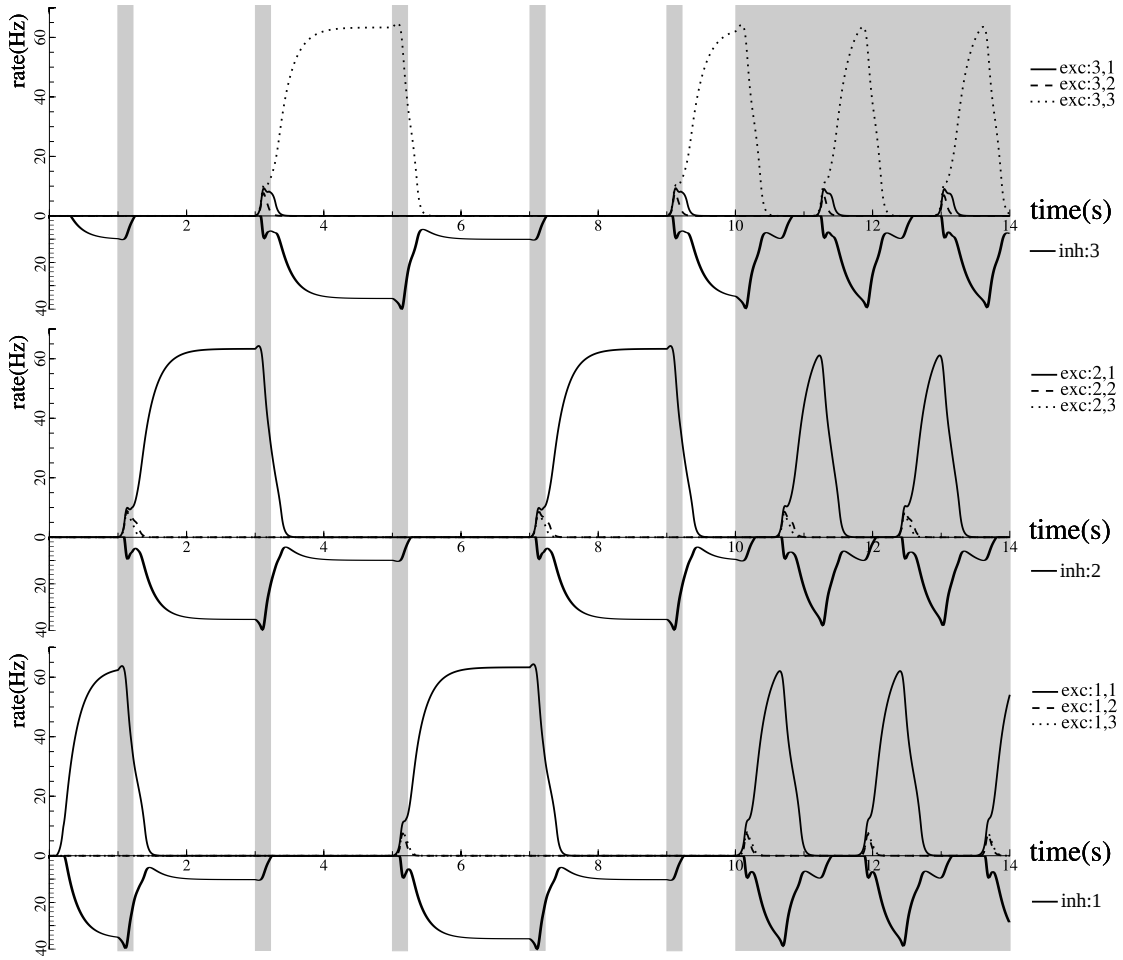


Figure 2.5: External background excitation was applied to all excitatory populations in the network of Figure 2.2 for 0.2s at 2s intervals starting from 1s, and then applied continuously starting from 10s. The plot shows the activity of excitatory populations in the network. The activities of inhibitory populations are plotted on reversed Y-axes. The application times of background excitation are shown as gray rectangles in the plot. When briefly applied, background excitation triggers a single transition in the sequence; When continuously applied, the network sequential activity is restored.

excitatory population in a WTA stage wins the competition, it is able to settle into an attractor state without triggering the next WTA stage. It will only provide a priming input to the next stage. An external diffuse background excitation will provide the necessary additional stimulation needed to activate the primed WTA stage and resume the sequence. Removal of this input will pause the sequence in its current state. This behavior is illustrated in Figure 2.5, where an external diffuse input is used to halt and resume the sequence. The attractors that materialize when the sequence is halted allow the network to effectively store the current state of the sequence (i.e., the stage at which the sequence has halted and the winning population of that stage).

Sequence Learning

If activity in a network of asymmetrically coupled WTA circuits like the one shown in Figure 2.2 is left to spontaneously develop, the network will eventually settle into a repeating sequential pattern. The sequential pattern is composed of a number of discrete transitions where each transition is characterized by the activity jumping from one WTA stage to the next. Each of the transitions forming this repeating pattern must obey the condition given by Eq. 2.2. Selective external input that targets some of the excitatory populations can dictate the winning population in each WTA circuit and override the sequence encoded in the feed-forward weights. After the input is removed, the externally-imposed sequential pattern may persist if its constituent transitions satisfy Eq. 2.2. But in general, this will not be the case. This is illustrated in Figure 2.6. After the external input is removed, the sequential activity is again controlled by the feed-forward excitatory weights and the final pattern is different from both the initial and the input-imposed sequences.

I introduce a simple rate-based plasticity rule that operates on the feed-forward excitatory-excitatory connections. The rule is given by:

$$\tau_w \dot{w}(t) = K u(t)([(v(t) - v_{th})]^+(w_{max} - w(t)) + [(v(t) - v_{th})]^-(w(t) - w_{min})) \quad (2.3)$$

$$[x]^+ = \max(0, \mathrm{x}) \quad , \quad [x]^- = \min(0, \mathrm{x})$$

where $w(t)$ is the connection weight, $u(t)$ the source population activity, and $v(t)$ the target population activity. K is a constant that controls the learning speed, and τ_w the learning time constant. This Hebbian plasticity rule is similar to the Bienenstock-Cooper-Munro(BCM) rule (Bienenstock et al., 1982). The main difference is that the threshold v_{th} that delimits the transition between potentiation and depression is a fixed constant. The sliding threshold used in the BCM rule is needed to keep the rule stable and avoid runaway potentiation of the weights. The plasticity rule given by Eq. 2.3 is trivially stable as the weight $w(t)$ is softly bound by w_{min} and w_{max}. The learning rule captures the dependence of potentiation and depression induction on the postsynaptic firing rate (Sjöström et al., 2001).

When an external input imposes a particular sequence on the network, this will lead to a reorganization of the weights of the feed-forward excitatory-excitatory connections. The feed-forward connections that are part of the input-imposed sequence, i.e., the connections that connect the winning excitatory population of one stage to the winning excitatory population of the subsequent stage, will potentiate. All connections going from the winning excitatory population of one stage to the losing excitatory populations of the next stage will depress. This is because activity in each excitatory populations will only go above v_{th} if it is the winning population. This is illustrated in Fig. 2.7. The K parameter in Eq. 2.3 was chosen so that a sequence has to propagate through the network only once to fully reorganize the feed-forward connection weights in a manner that favors the regeneration of the same sequence. By choosing a smaller value for K, we can reduce the rate at which the weights change and multiple iterations of the same sequence will then be needed to fully reorganize the weights. In Fig. 2.7, an initial external input reorganizes the plastic weights in order to store the sequence exc1,1 exc2,1 exc3,3. Later, another input imposes the sequence exc1,2 exc2,2 exc3,1 on the network, and in the process reorganizes the plastic weights in order to store the imposed sequence. The input-imposed sequence persists after input removal.

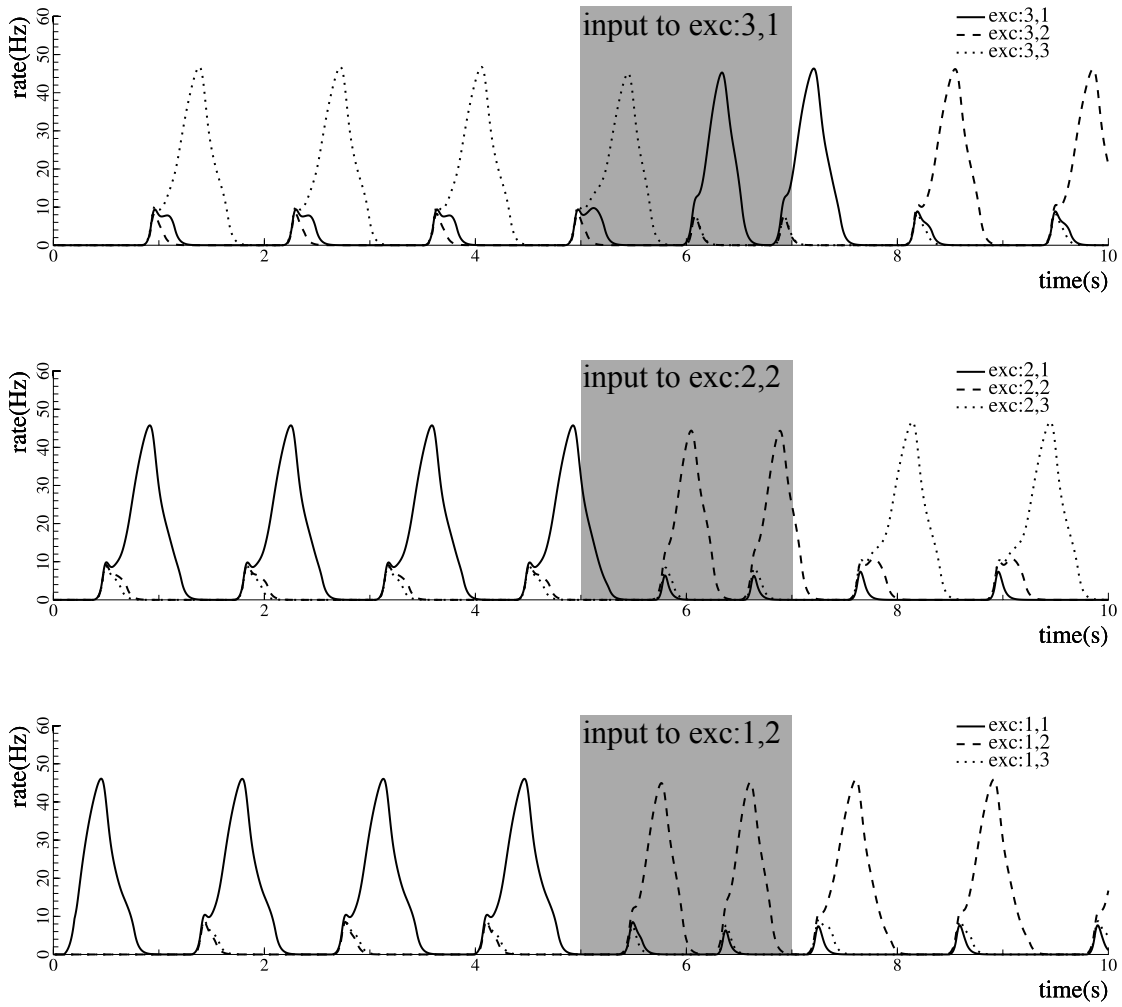


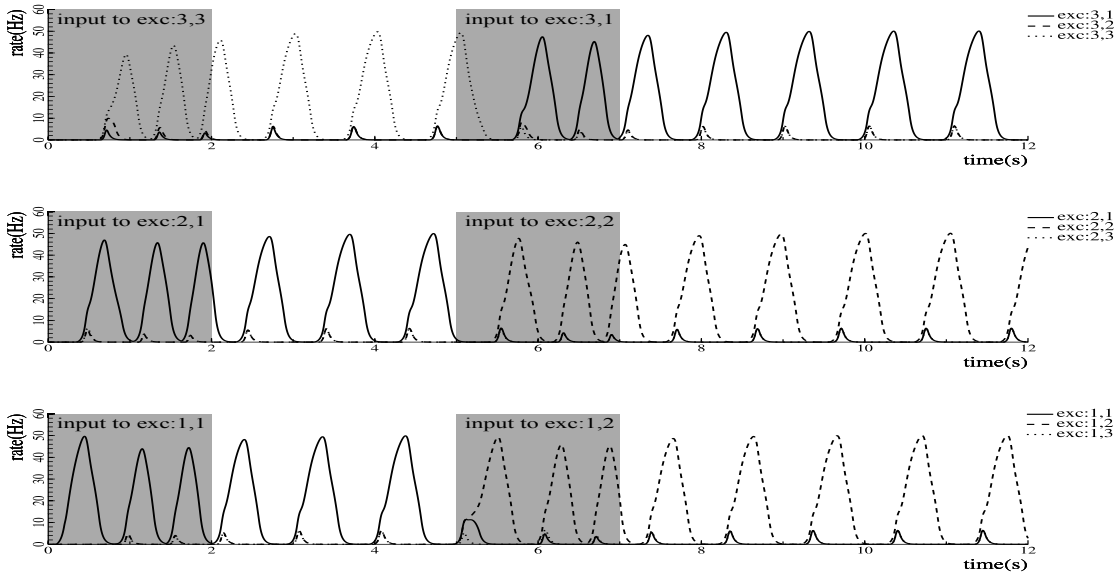
Figure 2.6: External input steering the path of the sequential activity. External excitation was applied between 5s and 7s to the populations exc:1,2, exc:2,2, and exc:3,1. External excitation application time is shown as gray rectangles on the plot. External excitation succeeds in steering the sequence but as soon as it is removed, the input-imposed pattern is lost.

2.4. Spiking Implementation of Asymmetrically Coupled Attractor Networks

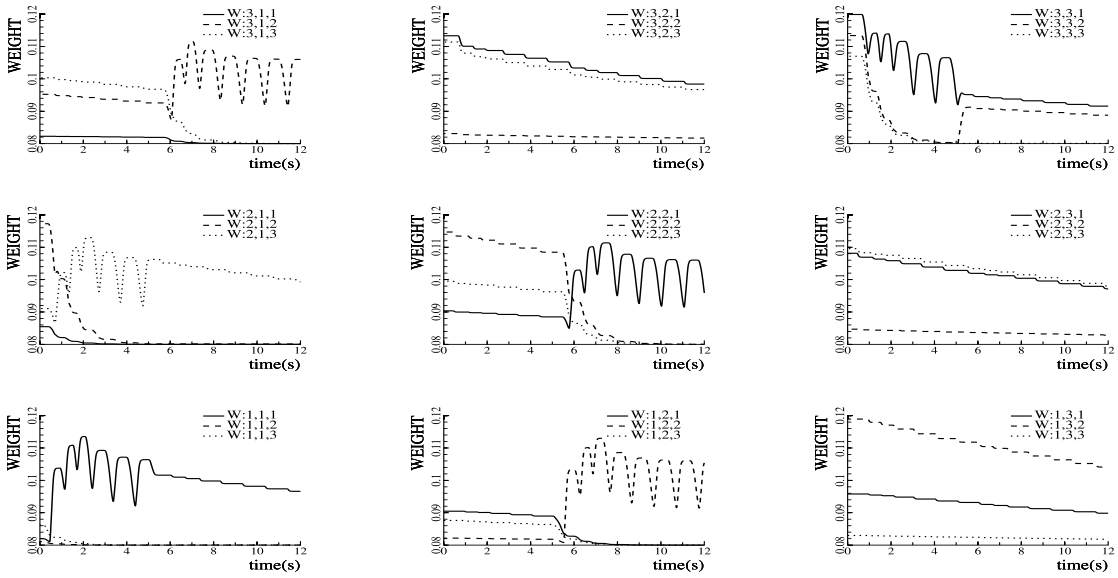
I implemented the network shown in Fig. 2.2 using populations of spiking integrate and fire neurons with conductance based synapses. Each neuron is modeled as:

$$C_m \dot{V}_m(t) = g_l(V_L - V_m(t)) + I_{AMPA} + I_{NMDA} + I_{GABA}$$
$$V_m(t) \leftarrow V_L \quad \text{if } V_m(t) > V_{firing} \tag{2.4}$$

$V_m(t)$ is the membrane potential, C_m the membrane capacitance. g_l is the leak conductance and V_L the resting potential. I_{AMPA}, I_{NMDA}, and I_{GABA} are the synaptic currents due to the activation of the $AMPA$, $NMDA$, and $GABA_A$ receptors respectively. The neuron fires when $V_m(t)$ crosses V_{firing}. $V_m(t)$ is then reset to V_L. After firing, the neuron enters a refractory period that lasts for T_{ref}. During the refractory period, the neuron is not integrating any synaptic



(a)



(b)

Figure 2.7: External input reorganizes the plastic feed-forward excitatory weights to store the imposed sequence. (a) Activity of the excitatory populations in the network. Gray rectangles indicate external excitation to specific populations. (b) The strengths of the feed-forward excitatory connections. W:i,j,k is the connection weight from population exc:i,j to population exc:i+1,k. The addition wraps around to stay in the range [1,3].

inputs.

I represent each excitatory population and each inhibitory population in the network shown in Fig. 2.2 by 30 excitatory neurons and 30 inhibitory neurons respectively. A connection between two populations is implemented by having each neuron in the source population form synapses on each neuron in the target population. The strength of each individual synapse is drawn from a random distribution. Appendix B contains the details of the synaptic models and the neuron and synapse parameters used in the following simulations.

Figure 2.8 shows the simulation results of the spiking implementation of the network shown in Fig. 2.2. Sequential activity was launched by providing external excitation to the population exc:1,1 during the initial 0.1s. The effect of small differences in the Gaussian distributed strengths of the feed-forward excitatory-excitatory synapses is amplified by the WTA mechanism. These small differences dictate the path of the sequential activity.

In Fig. 2.9, the inter-stage excitatory-excitatory connections have been weakened so that activity can not spontaneously propagate in the network. Uniform background excitation takes the form of a 300Hz Poisson spike train that activates $AMPA$-mediated conductances in all excitatory neurons. If background excitation is briefly applied, it triggers a single transition in the sequence. If it is continuously applied, sequential activity proceeds as in the spontaneous case shown in Fig. 2.8. As in the firing rate model, background excitation can be used to halt or resume the sequence. The sequential activity path is determined by differences in the feed-forward excitatory weights that were randomly chosen at the beginning of the simulation, and by the fluctuations in neural activity. The effect of the later can be seen when the population exc:1,3 is active. Due to fluctuations in the spiking pattern, activity in one case jumps to exc:2,1, and in others it jumps to exc:2,2. Also activity in population exc:3,1 jumps to different destinations.

I have shown in the context of the rate-based model that the network can learn sequences if the inter-stage feed-forward excitatory connections are plastic. For the spiking network, I introduce synaptic plasticity in the feed-forward synapses connecting the excitatory populations in one stage to the excitatory populations in the subsequent stage. I use a calcium-based biophysically realistic model of synaptic plasticity (Graupner & Brunel, 2012) in which pre-synaptic spikes and post-synaptic spikes trigger a calcium influx due to the activation of $NMDA$ receptors and the activation of voltage dependent calcium channels respectively. The calcium concentration in a synapse modulates its efficacy. Synaptic efficacy has bistable dynamics so that when there is no pre- or post-synaptic activity and the calcium concentration has decayed sufficiently, synaptic efficacy settles to one of two values: high or low. Appendix B contains more details about the synaptic plasticity rule.

Figure 2.9 shows the simulation results in a longer chain of 10 WTA stages. Each stage has 3 competing excitatory populations. External excitation that targets one excitatory population in each stage initially sets the path of the sequential activity and reorganizes the feed-forward plastic weights to store the input-imposed pattern. External excitation is a 150Hz Poisson spike train that activates $AMPA$-mediated conductances in the target populations. Between 1s and 2s, external excitation steers the sequential activity along a different path which reorganizes the existing pattern of feed-forward weights. Sequential activity thus proceeds along the new path after the removal of the input. External excitation also speeds up the propagation of activity as it puts the excitatory neurons closer to threshold and speeds up the winner selection process in each stage. Figure 2.9 also shows a sample of the efficacy of the feed-forward connections emanating from some excitatory populations. The efficacy of the feed-forward connection between two populations is the average of the efficacy of the 30^2 plastic connections that make up the all-to-all connectivity between the neurons in the two populations. I ran the simulation in Figure 2.9 a hundred times. Each time, the synaptic weights were randomly reinitialized and the Poisson external input recomputed. The network always learns the new input imposed sequence and

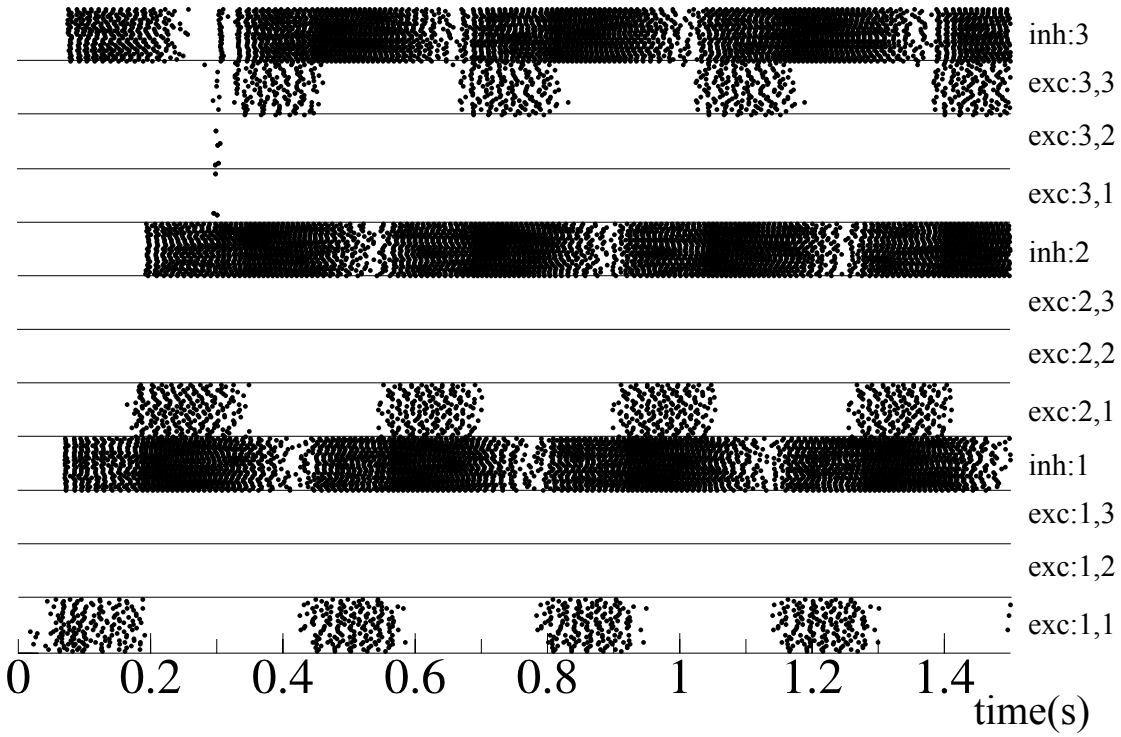


Figure 2.8: Spontaneous sequential activity in the network of Fig. 2.2 using spiking neurons. Each row is a raster plot of the activity of the 30 neurons making up a population.

reproduces it after the input is removed.

2.5. Self-timed Symbolic Computation

Some early hypotheses regard biological computation as symbolic in nature (Newell & Simon, 1976; Fodor, 1975; McCulloch & Pitts, 1943). However, there is no clear model that can explain in mechanistic terms how biologically-realistic networks may support large-scale symbolic computation or large scale finite state machines (FSMs). While there exist biologically-plausible neural implementations of FSMs (Rutishauser & Douglas, 2009; Neftci et al., 2013), these implementations do not scale well since a dedicated population is used to encode each state and each state transition. The one-hot encoding of the state space and the input space used in these networks lead to a linear increase of network size with the number of states and inputs.

A scalable neural architecture for carrying out symbolic computation, however, should employ a combinatorial or distributed representation scheme where the input space and state space sizes grow exponentially (not linearly) with the number of neural populations. The state transition model should also not be a lookup-table model where a distinct population is used to encode each transition, but rather the current state should be combined with input symbols (possibly from multiple sources) in successive stages (similar to successive gates in a digital processor) to produce a combinatorial/distributed representation of the next state and the output symbol. The FSMs implemented by such a neural architecture should also be insensitive to the delays in each processing stage and to the arrival times of input symbols, i.e, the neural architecture should be self-timed. This section describes a neural architecture that meets these requirements.

The networks described in this section make use of the asymmetric coupling scheme described in the previous sections to realize well defined patterns of sequential activity. These

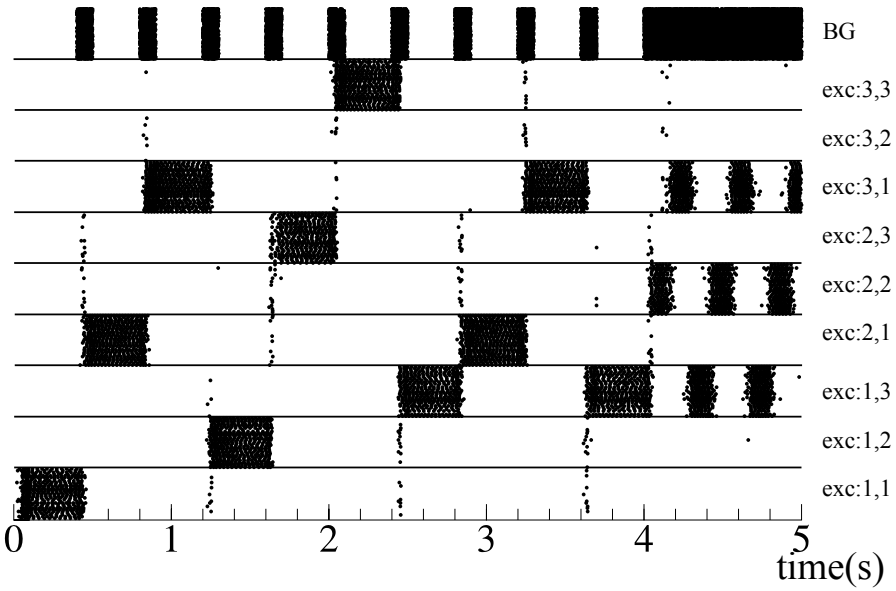


Figure 2.9: Raster plot of the 30 neurons in each excitatory population in the spiking version of the network in Fig. 2.2. Shown also is the external spike train, BG, that is used to excite all the excitatory populations. The excitatory-excitatory synapses in the feedforward paths were weakened in order to stop spontaneous propagation of activity. The external spike train, BG, triggers each step in the sequence.

input-dependent and state-dependent patterns are used to implement cascaded and state-dependent processing of input symbols. Throughout this section, I will only consider rate-based networks. The extension to spiking networks can be done in a similar way as in section 2.4. As a simple example, Fig. 2.11 shows the implementation of a binary 'AND' operation. By treating each WTA as one bit whose value is encoded as the identity of the winning population, one can easily see from the strength of the feedforward connections from A and B to C that for the four possible configurations of the winning populations in the two input WTAs (A and B), the winning population of WTA C encodes the logical 'AND' of the binary values of the two input WTAs. Since the feedforward connections from one input WTA are not strong enough to activate the target WTA, this 'AND' gate network only produces an output when its two inputs are available, i.e, when the two input WTAs are active. As soon as an output is produced, the activity in the output WTA shuts down the two input WTAs through the feedback excitatory-inhibitory connections.

Networks that implement cascaded processing using the asymmetric coupling scheme presented in this chapter, which is characterized by feedforward excitatory-excitatory connections and feedback excitatory-inhibitory connections, have an inherent 'flow control' mechanism. This is illustrated in the network in Fig. 2.12a which has four input bits: A, B, D, and F and which computes the function '((A AND B) OR D) and not(F)'. As shown in the simulation results in Fig. 2.12b, the first inputs to WTAs/bits A, B, D, and F arrive at 1s, 2s, 0.1s, and 4s respectively to yield the correct output in WTA/bit G. As in the previous example (Fig. 2.11), WTAs C, E, and G are activated only when all their inputs are available. The output produced by G is not 'consumed' by any subsequent processing stage. The next batch of inputs to A, B, D, and F arrives at 7s, 8s, 8s, and 6s respectively. These new inputs can not propagate to the output G, because the information in G from the previous batch of inputs has not yet been processed or consumed. WTAs/bits E and F are strongly inhibited by WTA G through feedback excitatory-inhibitory connections. Thus, F does not accept the input it receives at 6s, i.e, it does

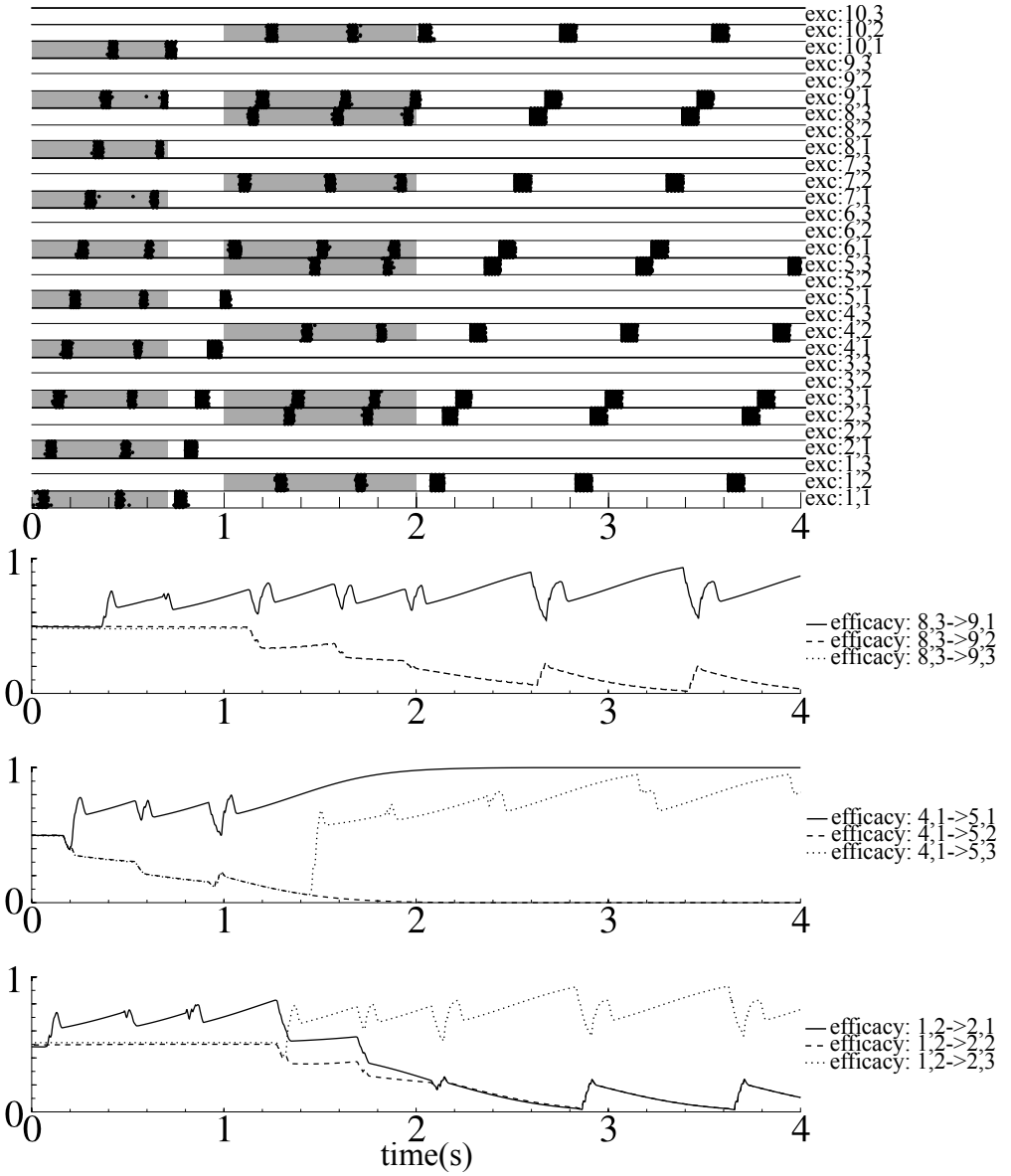
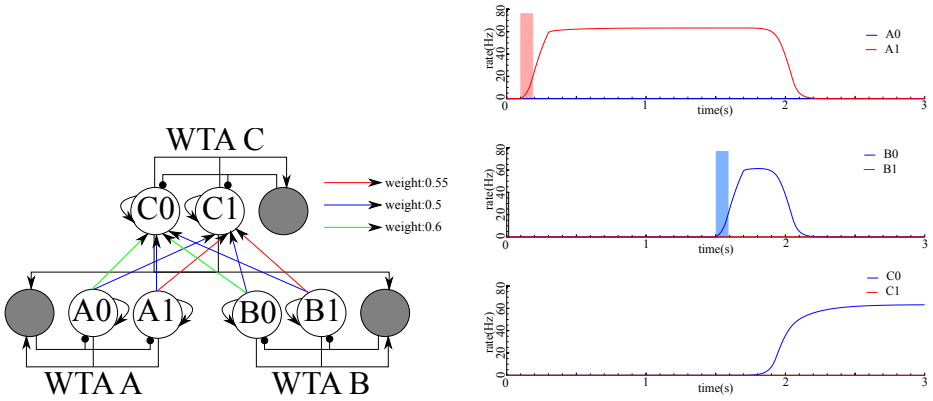


Figure 2.10: Sequence learning in a spiking network of 10 asymmetrically coupled WTA stages with 3 excitatory populations each and plastic feed-forward excitatory-excitatory synapses. Shown are the raster plots of the 30 excitatory populations. A gray rectangle on a population's raster plot indicates external input to that population. The average normalized efficacies of the feed-forward plastic connections emanating from 3 sample excitatory populations (exc:1,2, exc4,1, and exc:8,3) are shown.

not become active. Similarly, even though the two inputs to E (C and D) are active (for example at t=10s), E does not produce an output as it is being inhibited by G. The activity/value of C and D will thus remain constant (at an attractor) until the pipeline is 'unblocked' and E is disinhibited and able to generate an output. The pipeline will be unblocked when the value/activity of G is consumed/processed by some subsequent stage which, in the process, will shut down the activity in G and allow the new batch of inputs to propagate through the processing cascade.

Cascaded processing of information in chains of asymmetrically coupled attractor networks thus exhibit two important properties:



(a) (b)

Figure 2.11: Implementation of an 'AND' gate.(a) Network illustrating the convergence of two WTAs on a single WTA in an 'AND' operation where each WTA is treated as one bit. Inhibitory populations are shown in grey. All other populations are excitatory populations. The feedforward excitatory-excitatory connections from WTAs A and B to WTA C are shown as colored arrows. The shown weights of these connections/arrows are normalized relative to the weight needed to activate the target WTA (WTA C) when the source WTA activity is at its non-zero steady state. (b) Simulation results of the network in a. External input to WTAs A and B is shown as colored rectangles on their activity plots. WTA C is only activated when *both* its input WTAs (WTAs A and B) are active. The activity in WTA C then encodes the logical 'AND' of its inputs (in this case 0), then shuts down its input WTAs through the feedback excitatory-inhibitory connections shown in a

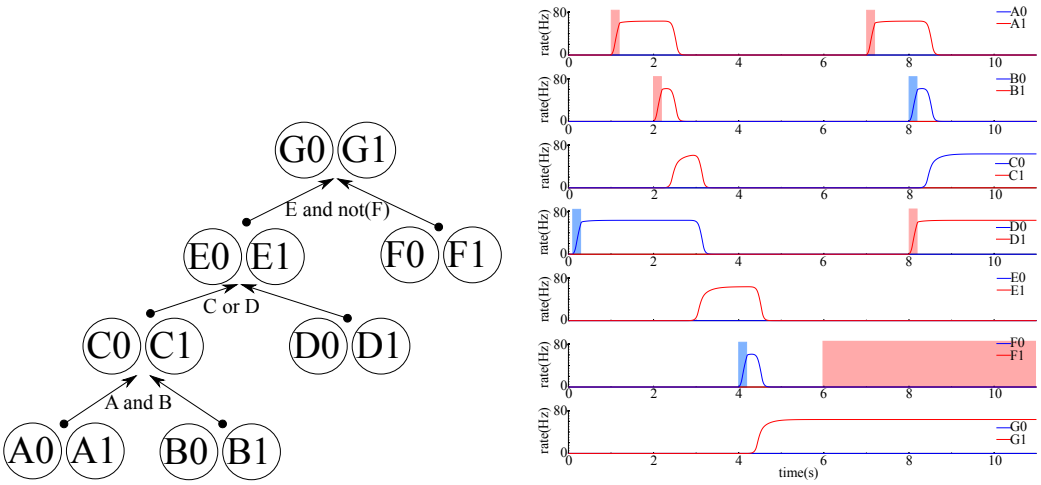
1. Each processing stage only generates an output or becomes active when all its inputs are available.

2. An active stage or a stage that is producing an output blocks all information streams feeding into it until its activity is processed by a subsequent stage

I will call a cascaded processing scheme that satisfies these two properties self-timed as the results of the processing cascade are independent of the delays in each stage, as well as the arrival times of the different inputs.

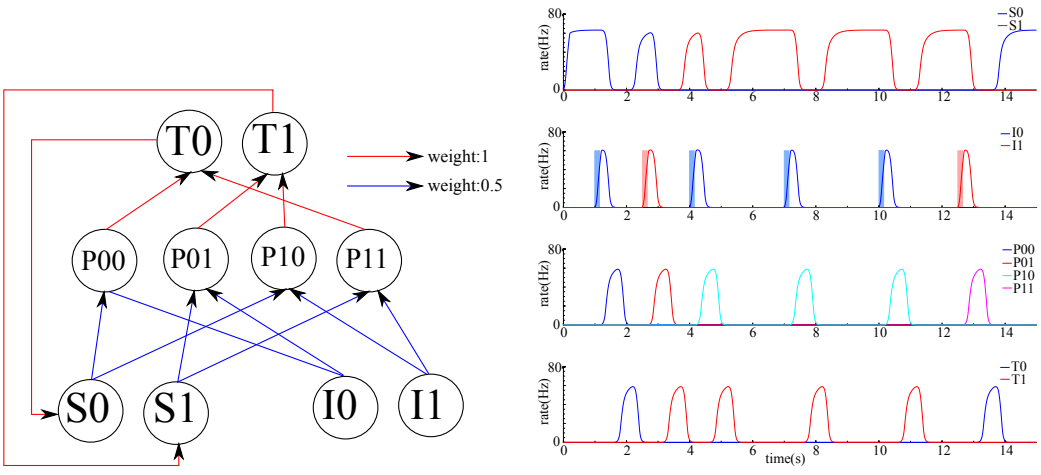
FSMs can be implemented by simply using a circular processing stream where output stages feed back into input stages. This is illustrated in Fig. 2.13 which shows a network that generates a running parity bit of the sequence of bits it receives through WTA I. The current parity is stored in WTA S. This parity bit is XORed with the incoming input bit to yield the new parity bit. Since the outputs of the XOR operation are not linearly separable in the input space (unlike the AND and OR operations which are implemented following the pattern in Fig. 2.11a), we first convert the two bits to a 4-valued representation, P, and then read out the XOR result in WTA T. WTA T then feeds back into WTA S to update the FSM state.

Since cascaded processing in the presented networks is self-timed, FSMs implemented using circular cascades in these networks will execute exactly one state transition for each complete set of input symbols. The FSM might have multiple inputs. Only when all the inputs are available, will the FSM execute a state transition. The self-timed property allows us to use a distributed representation of the state and input spaces where each input or state is represented by the activity in multiple WTAs. That is possible because the network behavior is insensitive to



(a) (b)

Figure 2.12: Illustration of the flow control mechanism. (a) Network computing the function '((A AND B) OR D) and not(F)' where the binary WTAs are treated as bits. The network is shown in simplified form where the inhibitory populations and the intra-WTA connections are not shown. The feedforward excitatory-excitatory connections between the WTAs are not explicitly shown. Instead, the logic function implemented by the feedforward connections are shown (again under the interpretation of a binary WTA as one bit). The feedback excitatory-inhibitory connections are also not explicitly shown but they follow the same pattern as in Fig. 2.11a. (b) Simulation results of the network in a. External inputs to the WTAs are shown as colored rectangles on their activity plots.



(a) (b)

Figure 2.13: Finite state machine implemented as a circular processing stream. (a) Network computing the parity of bits arriving through WTA I. The parity is stored in WTA S. Inhibitory populations, intra-WTA connections, and feedback excitatory-inhibitory connections are not shown. (b) Simulation of the network in a. External inputs to the WTAs are shown as colored rectangles on their activity plots.

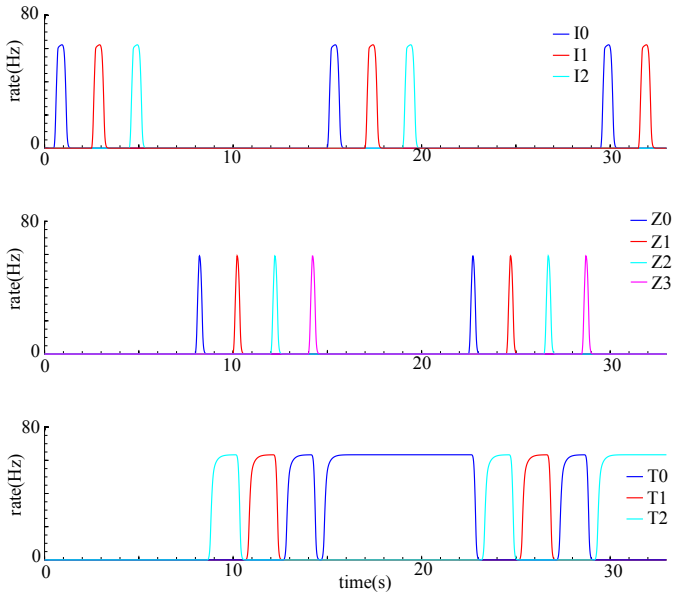


Figure 2.14: Simulation results of the network implementing the FSM F described in the main text when $W = 3$ and $D = 4$.

delays within the network as well as delays between the arrival times of the different parts of the input representation. The distributed representations of inputs and states will always combine in a well defined, delay-insensitive, manner to yield the new output and state representations.

To illustrate the advantages of a distributed state representation, consider a FSM, F, that can receive inputs through two WTAs : I and Z. The FSM receives a stream of symbols $I_1, I_2, ..$ through WTA I. Each of these can have W possible values, i.e, WTA I has W excitatory populations. At any point in time, the FSM might receive a symbol on WTA Z that can have D possible values: $0, .., D-1$. Assume $Z = j$, the FSM should then output the j^{th} symbol it received on WTA I counting backwards, i.e, if the FSM had received p symbols on WTA I, and it receives a symbol j on WTA Z, it should output symbol I_{p-j}. To be able to fulfill this task, it is clear the FSM has to store the last D symbols it received through WTA I since it may be asked to retrieve any of them.

This FSM can be implemented by a network with $6D + 3$ WTAs that are coupled in various asymmetric ways. $2D + 2$ of these WTAs have W excitatory populations, $3D$ have one excitatory population, and $D + 1$ have D excitatory populations for a total network size of $2DW + D^2 + 2W + 4D$ excitatory populations and $6D + 3$ inhibitory populations. Fig. 2.14 shows the simulation results of such a network when $W = 3$ and $D = 4$. As shown, the symbol arriving on WTA Z selects which of the last four symbols received on WTA I to output on WTA T. If population j in WTA Z is activated, then the network produces on WTA T the j^{th} symbol it received on WTA I counting backwards from the last received symbol.

An implementation of such a FSM using the scheme described in refs (Rutishauser & Douglas, 2009; Neftci et al., 2013) would have required two WTAs with W^D excitatory populations each to represent the state in a one-hot fashion, in addition to the populations implementing the state transitions. The distributed representation scheme used in the presented networks allows the state space to grow exponentially with the number of populations, therefore these networks can realize the described FSM much more efficiently (using only $2DW + D^2 + 2W + 4D$ excitatory populations)

2.6. Discussion

Networks whose dynamics are governed by fixed point attractors offer a powerful substrate for implementing a myriad of tasks such as associative memory (Hopfield, 1982), decision making (Wang, 2002; Soltani & Wang, 2006; Wang, 2008), and finite state machines (Rutishauser & Douglas, 2009). Stable network states can easily arise if activity in the network stabilizes at a high level due to saturating non-linearities in the network components. This, however, is not consistent with the general firing pattern of biological neurons which typically fire at rates far below the maximum rates set by the refractory mechanisms. In order for a network to exhibit non-trivial stable states at realistic activity levels, interacting inhibitory and excitatory neurons are needed (Amit & Brunel, 1997). The asymmetric coupling scheme used to couple WTA circuits in this chapter could thus be seen as a general scheme for coupling multiple circuits that individually exhibit stable attractors. The coupling scheme is characterized by feed-forward excitatory-excitatory connections from circuit A to circuit B and feedback excitatory-inhibitory connections from circuit B to circuit A where A and B are arbitrary neural circuits with stable fixed points of activity that arise out of the interplay between excitation and inhibition. The resulting network of coupled circuits will then exhibit well defined patterns of sequential activity.

One advantage of using a network that is composed of asymmetrically coupled attractor subnetworks in order to generate patterns of sequential activity is noise robustness; the state of each sub-network is always restored towards a quasi-attractor (Amit, 1992) when the sub-network is activated. We use the term quasi-attractor to denote a point in the state space of the network that exerts a pull on trajectories located in a certain region of the state space. When trajectories from this region approach the quasi-attractor, they are repelled towards another region of the phase space. This acts as a signal restoration mechanism that reduces the network's susceptibility to noise (see Fig. 2.4). Quasi-attractors are not fixed points of the network dynamics. This approach is therefore different from the one that is based on saddle points to guide sequential activity in winnerless competition networks (Rabinovich et al., 2008). In our case, the quasi-attractor point corresponds to the true attractor point that would be present in the uncoupled sub-network, that is however not present in the full network due to the asymmetric coupling connections.

If each attractor sub-network is a cooperative-competitive network(CCN) like the WTA, then there exist multiple competing quasi-attractor states in each of the cooperative - competitive sub-networks that make up the full network, and multiple patterns of sequential activity can be realized depending on which group of neurons wins the competition in each CCN sub-network, i.e., which quasi-attractor state is approached in each CCN sub-network. The number of sequences that can arise is exponential in the number of stages, or CCN subnetworks cascaded together. This distinguishes our approach from the synfire chain-like models of sequential activity. While synfire chains exhibit a moderate amount of noise robustness (Rotter & Aertsen, 1998), they do not exhibit stable attractors in which sequential activity can temporarily halt, nor is there a competition mechanism to enforce a categorical choice between a number of possible sequence paths.

Asymmetrically-coupled attractor networks can be naturally used to implement self-timed symbolic computation and FSMs as shown in section 2.5. One major advantage of such a scheme compared to previous neural implementations of FSMs such as (Rutishauser & Douglas, 2009; Neftci et al., 2013) is the invariance of the symbolic behavior of the network to network delays and to the arrival times of the input symbols. Another major advantage is the ability to represent inputs and states in a combinatorial or distributed manner. This leads to a vastly bigger state space (exponential in the number of populations in the network) compared to the networks in refs. (Rutishauser & Douglas, 2009; Neftci et al., 2013) whose state space size

only grows linearly with network size.

3

Winner-take-all Oscillators for Solving Constraint Satisfaction Problems

---

This chapter is based on the paper "Recurrent networks of winner-take-all oscillators for solving constraint satisfaction problems" by Hesham Mostafa and Lorenz K. Muller and Giacomo Indiveri, NIPS, 2013.

In this chapter, I investigate how neural networks can solve constraint satisfaction problems. These networks are fundamentally different from the symbolic computation network models presented in the previous chapter. Large-scale symbolic computation involves the cascaded processing of information and the notion of convergent and divergent processing streams. These notions are absent in the networks presented in this chapter and the next. These networks are composed of a large number of interacting units where each unit represents a variable or a constraint. There is no notion of input and output streams, rather, the units in the network mutually constrain each other and collectively contribute to the global behavior of the network, which is a search for maximally consistent states.

The processing of sensory information in the brain can be interpreted as the search for solutions to complex constraint satisfaction problems; sensory processing involves the integration of noisy and partial information from both sensory inputs and internal states to construct a consistent interpretation of the actual state of the environment. Consistency among different interpretations is likely to be inferred according to an internal model constructed from prior experience (Berkes et al., 2011). If we assume that a consistent interpretation is specified by a proper configuration of discrete variables, then it is possible to build an internal model by providing a set of constraints on the configurations that these variables are allowed to take. Searching for consistent interpretations under this internal model is equivalent to solving a max-constraint satisfaction problem (max-CSP).

Although there are many efficient algorithmic approaches to solving max-CSPs, it is still not clear how these algorithms can be implemented as biologically realistic dynamical systems. In particular, a challenging problem in systems whose dynamics embody a search for the optimal solution of a max-CSP is escaping from local optima. One possible approach is to formulate a stochastic neural network that samples from a probability distribution in which the correct solutions have higher probability (Habenschuss et al., 2013). However, the stochastic network will continuously explore the solution space and will not stabilize at fully consistent solutions. Another possible solution is to use simulated annealing techniques (Kirkpatrick et al., 1983). Simulated annealing techniques, however, cannot be easily mapped to plausible biological neural circuits due to the cooling schedule used to control the exploratory aspect of the search process. An alternative deterministic dynamical systems approach for solving combinatorial

optimization problems is to formulate a quadratic cost function for the problem and construct a Hopfield network whose Lyapunov function is this cost function (Hopfield & Tank, 1985). Considerable parameter tuning is needed to get such networks to converge to good solutions and to avoid local optima (Kamgar-Parsi, 1992). The addition of noise (Smith et al., 1998) or the inclusion of an initial chaotic exploratory phase (Chen & Aihara, 1995) in Hopfield networks partially mitigate the problem of getting stuck in local optima.

In this chapter and the next, I will present biologically-realistic neural networks whose dynamics execute a search for solutions to constraint satisfaction problems where the goal is to find a variable assignment that satisfies all the constraints encoded in the network connectivity. The model I present in this chapter is based on asymmetrically coupled WTA circuits, while the model presented in the next chapter is based on Gamma-band rhythmic inhibition. Both network models do not need a noise source to carry out the search process. Their deterministic dynamics directly realize a form of "usable computation" (Crutchfield, 1994) that is suitable for solving max-CSPs. The form of computation implemented is distributed and "executive-free" in the sense that there is no central controller managing the dynamics or the flow of information. The networks are cortically inspired as they are composed of coupled Winner-Take-All (WTA) circuits. The WTA circuit is a possible cortical circuit motif (Douglas & Martin, 2004) as its dynamics can explain the amplification of genico-cortical inputs that was observed in intracellular recordings in cat visual cortex (Douglas & Martin, 1992).

In addition to elucidating possible computational mechanisms in the brain, implementing "usable computation" with the dynamics of a neural network holds a number of advantages over conventional digital computation, including massive parallelism and fault tolerance. In particular, by following such dynamical systems approach, we can exploit the rich behavior of physical devices such as transistors to directly emulate these dynamics, and obtain more dense and power efficient computation (Mead, 1990). In chapter 7, I will describe how the dynamics of the networks presented in this chapter and the next can be simplified and used to implement efficient VLSI systems for solving large-scale constraint satisfaction problems.

3.1. Network Architecture

The basic building block of the proposed network is the WTA circuit in which multiple excitatory populations are competing through a common inhibitory population as shown in Fig. 3.1a. When the excitatory populations of the WTA network receive inputs of different amplitudes, their activity will increase and be amplified due to the recurrent excitatory connections. This will in turn activate the inhibitory population which will suppress activity in the excitatory populations until an equilibrium is reached. Typically, the excitatory population that receives the strongest external input is the only one that remains active (the network has selected a winner). By properly tuning the connection strengths, it is possible to configure the network so that it settles into a stable state of activity (or an attractor) that persists after input removal (Rutishauser et al., 2011).

The proposed network is a population-level, rate-based network. Each population is modeled as a linear threshold unit (LTU) which has the following dynamics:

$$\tau_i \dot{x}_i(t) + x_i(t) = \max(0, \sum_j w_{ji}(t) x_j(t) - T_i) \tag{3.1}$$

where $x_i(t)$ is the average firing rate in population i, $w_{ji}(t)$ is the connection weight from population j to population i, and τ_i and T_i are the time constant and the threshold of population i respectively.

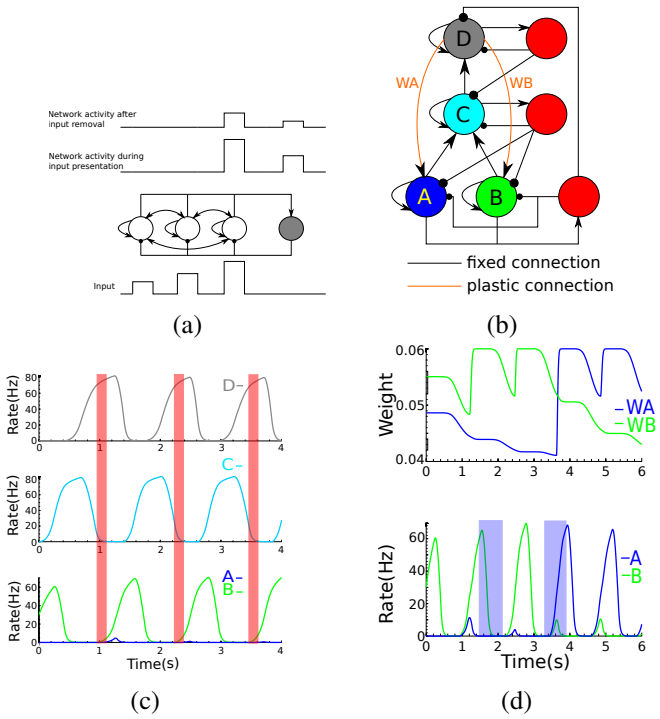


Figure 3.1: (a) A single WTA network. (b) Three coupled WTA circuits form the network representation of a single binary variable. Circles labeled A,B,C, and D are excitatory populations. Red circles on the right are inhibitory populations. (c) Simulation results of the network in (b) showing activity in the four excitatory populations. Shaded rectangles indicate the time intervals in which the state of the oscillator can be changed by external input. (d) Switching the state of the oscillator. The bottom plot shows the activity of the A and B populations. External input is applied to the A population in the time intervals denoted by the shaded rectangles. While the first input has no effect, the second input is applied at the right time and triggers a change in the variable/oscillator state. The top plot shows time evolution of the weights WA and WB.

The plastic connections in the proposed network obey a learning rule analogous to the Bienenstock-Cooper-Munro (BCM) rule (Bienenstock et al., 1982):

$$\dot{w}(t) = Ku(t)\left(\frac{(w(t)-w_{min})[v_{th}-v(t)]^{-}}{\tau_{dep}} + \frac{(w_{max}-w(t))[v(t)-v_{th}]^{+}}{\tau_{pot}}\right) \tag{3.2}$$

where $[x]^{+} = \max(0,x)$, and $[x]^{-} = \min(0,x)$. $w(t)$ is the connection weight, and $u(t)$ and $v(t)$ are the activities of the source and target populations respectively. The parameters w_{min} and w_{max} are soft bounds on the weight, τ_{dep} and τ_{pot} are the depression and potentiation time constants respectively, v_{th} is a threshold on the activity of the target population that delimits the transition between potentiation and depression, and K is a term that controls the overall speed of learning or the plasticity rate. The learning rule captures the dependence of potentiation and depression induction on the postsynaptic firing rate (Sjöström et al., 2001).

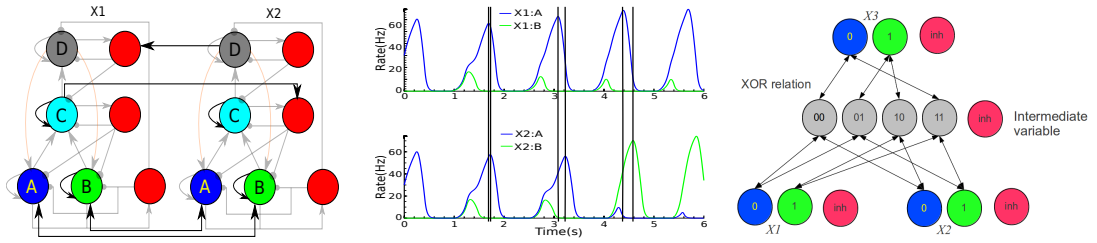
3.1.1. Variable Representation

Point attractor states in WTA networks like the one shown in Fig. 3.1a are computationally useful as they enable the network to disambiguate the inputs to the excitatory populations by making a categorical choice based on the relative strengths of these inputs. Point attractor dominated dynamics promote noise robustness at the expense of reduced input sensitivity: external input has to be large to move the network state out of the basin of attraction of one point attractor, and into the basin of attraction of another.

In this work, instead of using distinct point attractors to represent different variable values, we use limit cycle attractors. To obtain limit cycle attractors, we asymmetrically couple a number of WTA circuits to form a loop as shown in Fig. 3.1b. This has the effect of destroying the fixed point attractors in each WTA stage. As a consequence, persistent activity can no longer appear in a single WTA stage if there is no input. If we apply a short input pulse to the bottom WTA stage of Fig. 3.1b, we start oscillatory activity and we observe the following sequence of events: (1) the activity in the bottom WTA stage ramps up due to recurrent excitation, and when it is high enough it begins activating the middle WTA stage; (2) activity in the middle WTA stage ramps up and as activity in the inhibitory population of this stage rises, it shuts down the bottom stage activity; activity in the middle WTA stage keeps on increasing until it activates the top stage; (3) activity in the top WTA stage increases, shuts down the middle stage, and provides input back into the bottom stage via the plastic connections. As a consequence, a bump of activity continuously jumps from one WTA stage to the next. Since the stages are connected in a loop, the network will exhibit oscillatory activity. There are two stable limit cycles that the network trajectory can follow. The limit cycle chosen by the network depends on the outcome of the winner selection process in the bottom WTA stage. The limit cycles are stable as the weak coupling between the stages leaves the signal restoration properties of the destroyed attractors intact allowing activity in each WTA stage to be restored to a point close to that of the destroyed attractor. The winner selection process takes place at the beginning of each oscillation period in the bottom WTA stage. In the absence of external input, the dynamics of the winner selection process in the bottom stage will favor the population that receives the stronger projection weight from D. These projection weights obey the plasticity rule given by eq. 3.2.

The oscillatory network in Fig. 3.1b can represent one binary variable whose value is encoded in the identity of the winning population in the bottom WTA stage, which determines the limit cycle the network follows. The identity of the winning population is a reflection of the relative strengths of WA and WB. More than two values can be encoded by increasing the number of excitatory populations in the bottom WTA stage. Fig. 3.1c shows the simulation results of the network in Fig. 3.1b when the weight WB is larger than WA. This is expressed by a limit cycle in which populations B,C, and D are periodically activated.

During the winner selection process in the bottom WTA stage, the WTA circuit is very sensitive to external input, which can bias the competition towards a particular limit cycle. Once the winner selection process is complete, i.e, activity in the winning population has ramped up to a high level, the WTA circuit is relatively insensitive to external input. This is illustrated in Fig. 3.1d, where input is applied in two different intervals. The first external input to population A arrives after the winner, B, has already been selected so it is ineffective. A second external input having the same strength and duration as the first input arrives during the winner selection phase and biases the competition towards A. As soon as A wins, the plasticity rule in eq. 3.2 causes WA to potentiate and WB to depress so that activity in the network continues to follow the new limit cycle even after the input is removed.



(a) (b) (c)

Figure 3.2: (a) Coupling $X1$ and $X2$ to implement the constraint $X1 \neq X2$. (b) Activity in the A and B populations of $X1$ and $X2$ that are coupled as shown in (a). (c) Constraint involving three variables: $X1\,XOR\,X2 = X3$. Only the bottom WTA stages of the four variables and the inter-variable connections coupling the bottom WTA stages are shown.

3.1.2. Constraint Representation

Each variable, as represented by the network in Fig. 3.1b, is a multi-stable oscillator. Pair-wise constraints can be implemented by coupling the excitatory populations of the bottom WTA stages of two variables. Fig 3.2a shows the implementation of a constraint that requires two variables to be unequal, i.e., one variable should oscillate in the cycle involving the A population, and the other in the cycle involving the B population. Variable $X1$ will maximally affect $X2$ when the activity peak in the bottom WTA stage of $X1$ coincides with the winner selection interval of $X2$ and vice versa. The coupling of the middle and top WTA stages of the two variables in Fig. 3.2a is not related to the constraint, but it is there to prevent coupled variables in large networks from phase locking. We explain why this is important in the next section. We define the zero phase point of a variable as the point at which activity in the winning excitatory population in the bottom WTA stage reaches a peak and we assume the phase changes linearly during an oscillation period (from one peak to the next). The phase difference between two coupled variables determines the direction and strength of mutual influence. This can be seen in Fig. 3.2b. Initially the constraint is violated as both variables are oscillating in the A cycle. $X1$ gradually begins to lead $X2$ until at a particular phase difference, input from $X1$ is able to bias the competition in $X2$ so that the B population in $X2$ wins even though the A population is receiving a stronger projection from the D population in $X2$.

A constraint involving more than two variables can be implemented by introducing an intermediate variable which will in general have a higher cardinality than the variables in the constraint (the cardinality of a variable is reflected in the number of excitatory populations in the bottom WTA stage; the middle and top WTA stages have the same structure irrespective of cardinality). An example is shown in Fig. 3.2c where three binary variables are related by an XOR relation and the intermediate variable has four possible states. The tertiary XOR constraint has been effectively broken down into three pair-wise constraints. The only states, or oscillatory modes, of $X1$, $X2$, and $X3$ that are stable under arbitrary phase relations with the intermediate variable are the states which satisfy the constraint $X1\ XOR\ X2 = X3$.

3.2. Oscillations and the Modulation of Effective Connectivity

From simulations, we observe that the phase differences between the variables/oscillators are highly irregular in large networks comprised of many variables and constraints. These irregular phase relations enable the network to search for the optimal solution to a max-CSP. The weight attached to a constraint is an analogue quantity that is a function of the phase differences between the variables in the constraint. The phase differences also determine which of the variables in a violated constraint changes in order to satisfy the constraint (see Fig. 3.2b). The irregular phase relations result in a continuous perturbation of the strengths of the different constraints by modulating the effective network connectivity embodying these constraints. This is what allows the network to escape from the local optima of the underlying max-CSP. At a local optimum, the ongoing perturbation of constraint strengths will eventually lead to a configuration that de-emphasizes the currently satisfied constraints and emphasizes the unsatisfied constraints. The transiently dominant unsatisfied constraints will reassign the values of the variables in their domain and pull the network out of the local optimum. The network thus searches for optimal solutions by effectively perturbing the underlying max-CSP. Under this search scheme, states that satisfy all constraints are dynamically stable since any perturbation of the strengths of the constraints defining the max-CSP will result in a constraints configuration that reinforces the current fully consistent state of the network.

In principle, if some variables/oscillators phase-lock, then the weights of the constraint(s) among these variables will not change anymore, which will impact the ability of the network to find good solutions. In practice, however, we see that this happens only in very small networks, and not in large ones, such as the networks described in the following sections.

3.3. The Non-stochastic Search Scheme

We simulated a recurrent neuronal network that represents a CSP that has ten binary variables and nine tertiary constraints (see Fig. 3.3a). Each variable is represented by the network in Fig. 3.1b. Each tertiary constraint is implemented by introducing an intermediate variable and using a coupling scheme similar to the one in Fig. 3.2c. We constructed the problem so that only two variable assignments are fully consistent. The problem is thus at the boundary between over-constrained and under-constrained problems which makes it difficult for a search algorithm to find the optimum (Kanefsky & Taylor, 1991).

We ran 1000 trials starting from random values for the synaptic weights within each variable (each variable effectively starts with a random value). The network always converges to one of the optimal variable assignments. Fig. 3.3b shows a histogram of the number of oscillation cycles needed to converge to an optimal solution in the 1000 trials. The number of cycles is averaged over the ten variables as the number of cycles needed to converge to an optimal solution is not the same for all variables. Although the sub-networks representing the variables are identical, each oscillates at a different instantaneous frequency due to the non-uniform coupling and switching dynamics. Fig. 3.3c shows how the network state evolves in a sample trial. Due to the continuous perturbation of the weights caused by the irregular phase relations between the variables/oscillators, the network sometimes takes steps that lead to the violation of more constraints. This prevents the network from getting stuck in local optima.

We model the arrival of external evidence by activating an additional variable/oscillator that has only one state, or limit cycle, and which is coupled to one of the original problem variables.

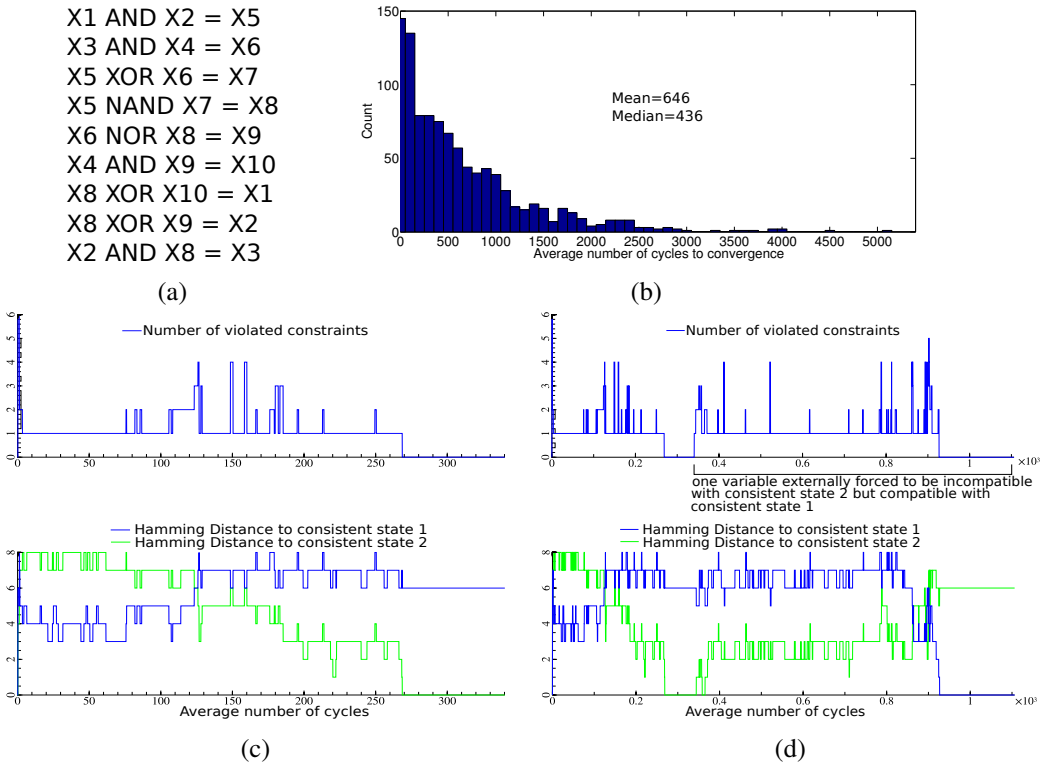


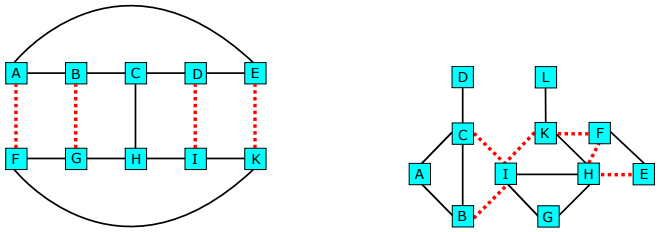
Figure 3.3: Solving a CSP with ten binary variables and nine tertiary constraints. (a) CSP definition. (b) Histogram of the number of cycles needed for convergence, averaged over all ten variables, in 1000 trials. (c) Evolution of network state in a sample trial. The top plot shows the number of constraints violated by the variable assignment decoded from the network state. The bottom plot shows the Hamming distance between the decoded variable assignment to each of the two fully consistent solutions. (d) One variable is externally forced to take a value that is incompatible with the current fully consistent variable assignment. The search resumes to find a fully consistent variable assignment that is compatible with the external input.

External evidence in this case is sparse since it only affects one problem variable. External evidence also does not completely fix the value of that one problem variable, but rather, the single state "evidence variable" affects the problem variable only at particular phase differences between the two. Fig. 3.3d shows that the network is able to take the external evidence into account by searching for, and finally settling into, the only remaining fully consistent state that accommodates the external evidence.

3.4. Sampling Interpretation

This section was done in collaboration with Lorenz Muller.

As shown in the previous section, if a fully consistent solution exists, the network state will end up in that solution and stay there. If no such solution exists, the network will never settle into one variable assignment, but will keep exploring possible assignments and will spend more time in solutions that satisfy more constraints. This behavior can be interpreted as a sampling process where each oscillation cycle lets one variable re-sample its current state; at any point



(a) (b)

Figure 3.4: Ising model type problems. Each square indicates a binary variable like in Fig. 3.1b; solid black lines denote a constraint requiring two variables to be equal, dashed red lines a constraint that requires two variables to be unequal. In both problems, all states violate at least one constraint.

in time, the network state represents a sample from a probability distribution defined over the space of all possible solutions to the max-CSP, where more consistent solutions have higher probability. The oscillatory dynamics thus give rise to a decentralized, deterministic, and time-continuous sampling process. This sampling analogy is only valid when there are no fully consistent solutions. To illustrate this behavior, we consider two max-CSPs having an Ising-model like structure as shown in Figs. 3.4a, 3.4b. We describe the behavior of two networks that represent the max-CSPs embodied by these two graphs.

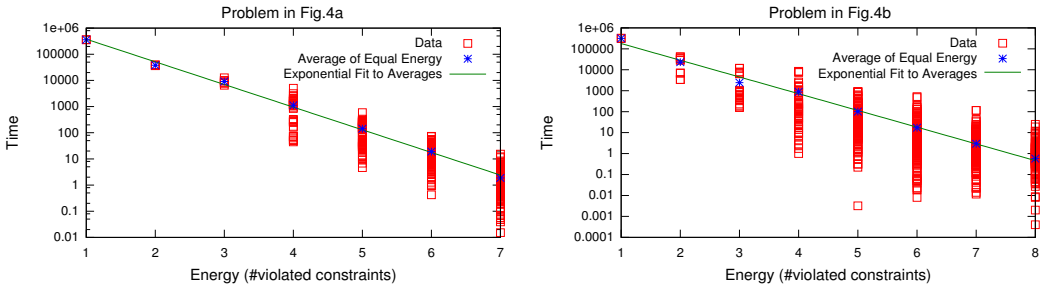
Let $E(s)$ be a function that maps a network state s to the number of constraints it violates; this is analogous to an energy function and we will refer to $E(s)$ as the energy of state s. For the problem in Fig. 3.4a, we observe that the average time the network spends in states with energy E is $t(E) = c_1 \exp(-c_2 E)$ as can be seen in Fig. 3.5a. The network spends almost equal times in complementary states that have low energy. Complementary states are maximally different but the network is able to traverse the space of intervening states, which can have higher energy, in order to visit the complementary states almost equally often.

We expect the network to spend less time in less consistent states; the higher the number of violated constraints, the more rapidly the variable values change because there are more possible phase relations that can emphasize a violated constraint. However, we do not have an analytical explanation for the good exponential fit to the energy-time spent relation. We expect a worse fit for high energies. For example, the network can never go into states where all constraints are violated even though they have finite energies.

For the problem in Fig. 3.4b, not all states of equally low energy are equally likely as can be seen in Fig. 3.5b. For example, the states of energy 2, where C and D (or K and L) are unequal, are less likely than other assignments of the same energy. This is not surprising. When C is in some state, D has no reason to be in a different state (no other variables try to force it to be different from C) apart from the memory in its plastic weights. We expect that this effect becomes small for sufficiently densely connected constraint graphs. The exponential fit to the averages is still very good in Fig. 3.5b.

3.5. Discussion

Oscillations are ubiquitous in cortex. Local field potential measurements as well as intracellular recordings point to a plethora of oscillatory dynamics operating in many distinct frequency bands (Wang, 2010). One possible functional role for oscillatory activity is that it rhythmically modulates the sensitivity of neuronal circuits to external influence (Fries, 2005; Womelsdorf



(a) (b)

Figure 3.5: Behavior of two networks representing the CSPs in Fig. 4. Red squares are data points (the time the network spent in one particular state), a blue star is the average time spent in states of equal energy and the green line is an exponential fit to the blue stars. (a) Note that at energies 1 and 2 there are two complementary states *each* that are visited almost equally often. (b) Not all assignments of energy 2 are equally probable in this case (not a finite samples artifact, but systematic) as can be seen in the bimodal distribution there. This is caused by variables that are part of only one constraint.

et al., 2007). Attending to a periodic stimulus has been shown to result in the entrainment of delta-band oscillations (2-4 Hz) so that intervals of high excitability coincide with relevant events in the stimulus (Lakatos et al., 2008). I have used the idea of oscillatory modulation of sensitivity to construct multi-stable neural oscillators whose state, or limit cycle, can be changed by external inputs only in narrow, periodically recurring temporal windows. Selection between multiple limit cycles is done through competitive dynamics which are thought to underlie many cognitive processes such as decision making in prefrontal cortex (Wang, 2008).

External input to the network can be interpreted as an additional constraint that immediately affects the search for maximally consistent states. Continuous reformulation of the problem, by adding new constraints, is problematic for any approach that works by having an initial exploratory phase that slowly morphs into a greedy search for optimal solutions, as the exploratory phase has to be restarted after a change in the problem. For a biological system that has to deal with a continuously changing set of constraints, the search algorithm should not exhibit an exploratory/greedy behavior dichotomy. The search procedure used in the proposed networks does not exhibit this dichotomy. The search is driven solely by the violated constraints. This can be seen in the sampling-like behavior in Fig. 3.5 where the network spends less time in states that violate more constraints.

In the next chapter, I will investigate a related network model that uses a model of Gamma-band rhythmic inhibition to search for CSP solutions. The behavior of this network model, unlike the networks presented in this chapter, can be investigated analytically. The networks presented in the next chapter highlight a novel computational role for Gamma oscillations, and generate testable predictions that can be used to verify whether Gamma-band neural oscillations do indeed underlie the search for maximally consistent input interpretations in biological networks.

4

Gamma-band Rhythmic Inhibition and the Search for Maximally Consistent States

---

This chapter is based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa and Lorenz K. Muller and Giacomo Indiveri, Neural Computation, in press.

Gamma-band rhythmic inhibition is a ubiquitous phenomenon in neural circuits, yet its computational role still remains elusive. In this chapter, I show that a model of Gamma-band rhythmic inhibition allows networks of coupled cortical circuit motifs to search for network configurations that best reconcile external inputs with an internal consistency model encoded in the network connectivity. I show that Hebbian plasticity allows the networks to learn the consistency model by example. The search dynamics driven by rhythmic inhibition enable the described networks to solve difficult constraint satisfaction problems without making assumptions about the form of stochastic fluctuations in the network. These search dynamics are well approximated by a stochastic sampling process. The described networks can reproduce perceptual multi-stability phenomena with switching times that are a good match to experimental data and they provide a general neural framework which can be used to model other 'perceptual inference' phenomena.

Gamma oscillations are rhythmic patterns of neural activity in the 30-80 Hz frequency range that have been measured in the extracellular fields of multiple brain areas across many species (Buzsáki & Wang, 2012). They have been associated with attention (Womelsdorf et al., 2007), working memory (Deco & Rolls, 2003), and the execution of cognitive tasks (Tallon-Baudry et al., 1997). Local rhythmic inhibition is a fundamental feature of Gamma oscillations (Stein & Sarnthein, 2000; Buzsáki & Draguhn, 2004) that acts to modulate the firing rate of the local circuit as well as its sensitivity to external input (Cardin et al., 2009). Although many studies have elucidated several aspects of the biophysical mechanisms underlying Gamma-band rhythmic inhibition (Jadi & Sejnowski, 2014; Brunel & Wang, 2003), its computational and functional role is still a matter of debate (Buzsáki & Wang, 2012; Jadi & Sejnowski, 2014).

In this chapter, I investigate the functional role of local Gamma-band rhythmic inhibition in rate-based networks of neurons configured as coupled Winner-Take-All (WTA) circuits. The WTA circuits are used as models of local cortical circuit motifs (Douglas & Martin, 2004). The strength of the effective connectivity between a pair of coupled WTAs is continuously modulated by the phase difference between their local rhythms. Higher coherence between these rhythms leads to more reliable communication between the WTA pair in line with the 'communication through coherence' hypothesis (Bosman et al., 2012; Fries, 2005). On a global level, I show that oscillatory inhibition drastically alters the dynamics of coupled WTA networks and

allows them to search for activity states that best satisfy the constraints encoded in the network connectivity while being consistent with the externally applied inputs. I show that rhythmic inhibition effectively allows neural networks to solve constraint satisfaction problems. This result is particularly relevant to neural models of sensory processing. A long theoretical tradition casts sensory processing as being a process of inferring the maximally consistent interpretations of imperfect sensory input where consistency is judged according to an internal model of the environment constructed from prior experience (von Helmholtz & Southall, 1925; Friston, 2003; Berkes et al., 2011). There are two fundamental questions that arise when considering the neural substrate underlying the search for consistent interpretations: (1) how to *learn and represent* the consistency model; (2) how to use the consistency model to obtain *plausible interpretations* of ambiguous inputs. The networks I describe address these two questions within a biologically realistic framework.

The networks I describe in this chapter never get stuck at non-optimal input interpretations; if a fully consistent interpretation of an unambiguous input exists, the network will find and stabilize at that interpretation; if the input is ambiguous or the set of constraints irreconcilable, the network will continuously switch between the most consistent states or input interpretations. In the latter case, I show that the network trajectory can be well approximated by a stochastic Markov Chain Monte Carlo (MCMC) sampler, which is remarkable considering that the network trajectory is deterministic. This continuous switching between equally consistent interpretations can be used to model perceptual multi-stability phenomena. Unlike the majority of previous models, though, the initially 'naive' network can learn by example (through Hebbian plasticity in the network connections) what constitutes consistent inputs. The multi-stability phenomenon emerges when the network receives ambiguous input that does not admit a consistent interpretation according to the previously seen examples. The presented network architecture can be used as a biologically motivated neural substrate on which to ground various 'perception as inference' theories.

4.1. Modeling Assumptions

The networks I present are composed of a number of local neural circuits that interact through long range excitatory connections and that each receive local oscillatory inhibition. The properties of the oscillatory inhibition model the salient features of the rhythmic inhibition that underlies Gamma oscillations. The following are biological justifications for the various modeling assumptions I use:

1. *WTA circuits are local neural circuit motifs*: WTA circuits are a fairly realistic model of local cortical circuitry and their behavior is well understood (Rutishauser et al., 2012). They are considered a potential cortical circuit motif (Douglas & Martin, 2004).

2. *Oscillatory inhibition is local to each WTA*: Gamma oscillations typically have a local origin (Buzsáki & Wang, 2012; Stein & Sarnthein, 2000; Ainsworth et al., 2011). Gamma oscillations in local field potential typically arise from the rhythmic firing of basket cells that have predominantly local arborization (Fries, 2009). It is a general phenomenon that faster rhythms tend to develop locally (Buzsáki & Draguhn, 2004). We do not model the biophysical mechanisms underlying Gamma rhythm generation. Instead, we directly impose a local oscillatory inhibition waveform on all populations of the local circuit.

3. *Local oscillatory inhibition is strong enough to shut down the activity in the local circuit*: There is strong evidence for the involvement of interneurons containing the calcium binding protein Parvalbumin in the various types of oscillatory discharges underlying Gamma

oscillations (Sohal et al., 2009; Cardin et al., 2009; Tiesinga & Sejnowski, 2009). Interneurons expressing Parvalbumin, such as basket cells and chandelier cells, mainly target the soma, the axon initial segment, and the proximal dendrites of the excitatory principal cells (Markram et al., 2004) making them particularly effective in rhythmically silencing their target neurons.

4. *The local oscillatory inhibition waveforms have different frequencies*: There is evidence that Gamma oscillations recorded from even nearby regions in the same cortical area have significantly different frequencies (Ray & Maunsell, 2010). Phase relations between Gamma oscillations recorded from nearby points in the cortex are continuously changing (Womelsdorf et al., 2007; Fries, 2009), and the phases of local cortical Gamma rhythms vary in an irregular manner (Burns et al., 2010). Having different oscillation frequencies is one simple way to obtain continuously changing phase relations. The different local rhythms can be highly coherent and we investigate the effect of this coherence on the model behavior. The only requirement of our model is that the space of possible phase relations is continuously explored with no perfect and persistent phase lock between any of the local rhythms.

4.2. Network Architecture

The networks presented in this chapter are rate-based, population-level networks of interacting WTA circuits. Each WTA circuit undergoes local oscillatory inhibition. Each neural population in a WTA is modeled as a linear threshold unit (LTU) following a mean-field approach (Amit & Brunel, 1997; Knight, 2000; Mattia & Del Giudice, 2002): the mean firing rate of a population of neurons is represented by a dynamical variable that obeys the following equation:

$$\tau_i \frac{d}{dt} x_i(t) + x_i(t) = f\left(\sum_j w_{ji}^{fast} x_j(t) + \sum_j w_{ji}^{slow}(t) s_j(t) - T_i - H_i(t) \right) \tag{4.1a}$$

$$\tau_{\text{NMDA}} \frac{d}{dt} s_j(t) + s_j(t) = x_j(t) \tag{4.1b}$$

where $x_i(t)$ represents the average firing rate of population i, τ_i is the population time constant, T_i is the population threshold. The term $H_i(t)$ denotes the rhythmic inhibition received by population i, which, for simplicity, we model throughout as a periodic rectangular function. The function $f(x) = \max(0, x)$ is the linear threshold function which has been shown to be a good approximation of the steady state population firing rate in mean-field models (Fourcaud & Brunel, 2002; Fusi & Mattia, 1999). The synaptic current injected into population i due to the activity of population j has a slow component and a fast component. This division attempts to capture the effects of the different time constants of the synaptic currents mediated by NMDA receptors on one hand, and AMPA and GABA_A receptors on the other hand (Koch, 1999). The fast current component is modeled by scaling the instantaneous activity of source population j by the weight w_{ji}^{fast}, which can be positive to represent the contributions from AMPA-type synapses, or negative to represent the contributions from GABA_A-type synapses. The slow current component is modeled by low-pass filtering the activity of source population j with a time constant τ_{NMDA} to yield the variable $s_j(t)$, and then scaling $s_j(t)$ by the weight w_{ji}^{slow}. This models the contributions from NMDA synapses. All connections originating from excitatory populations have an AMPA/fast and an NMDA/slow component. Equation 4.1 describes the

behavior of a generic excitatory or inhibitory population. The first order dynamics of Eq. 4.1 is a simplification of the transient, or non-steady state, behavior of a population of spiking neurons (Mattia & Del Giudice, 2002; Knight, 2000). Our results, however, do not depend on the exact time course of the evolution of activity in the WTA circuits provided a clear winner is selected in each inhibition cycle.

Parameter values used in the simulations together with the full system of equations describing the proposed networks are given in appendix C. The parameters in each WTA circuit were chosen so that the WTA can have a number of non-zero stable attractor states when oscillatory inhibition is off (low). The number of attractor states is the same as the number of excitatory populations. The threshold of the excitatory population is negative to model background activity that pushes the WTA towards one of the non-zero attractor states when oscillatory inhibition is off. When oscillatory inhibition is on, all excitatory populations are silenced. The coupling connections between two WTA circuits enable activity in one WTA circuit to bias the winner selection process in the other WTA circuit but not to change the identity of the winning population once that population activity has ramped up beyond the initial winner selection phase. The inter-WTA coupling connections are also not strong enough to enable one WTA circuit to induce activity in another WTA circuit that is shut down by the oscillatory inhibition.

The network architecture and example simulation results are shown in Fig. 4.1. When the oscillatory inhibition targeting a WTA is in its off phase, the excitatory population receiving the largest external input wins and suppresses the activity in the other excitatory populations in the WTA. The NMDA-mediated recurrent synaptic current has a time constant that is longer than the oscillatory inhibition period. Since this current is always larger in the winning population, it is able to bias the competition so that the winning population keeps on winning in subsequent inhibition cycles after the external input is removed as shown in Fig. 4.1c.

Figure 4.1d shows two coupled WTA circuits in which the coupling connections encode a consistency condition. Each WTA circuit represents a discrete variable with two states A and B. The inter-WTA coupling connections dictate that when the population representing state A is winning in one WTA circuit, then the population representing state B should be winning in the other. The activity of one WTA circuit, however, is able to influence the state of the other WTA only when its interval of heightened activity coincides with the winner selection interval (the interval just after oscillatory inhibition switches off) in the other WTA circuit. This effect is illustrated in Fig. 4.1e where we choose slightly different frequencies for the oscillatory inhibition in the two WTA circuits.

This example shows how the effective strength of the inter-WTA coupling connections depends on the phase difference between their rhythmic inhibition cycles. This is compatible with the hypothesis that the effective strength of the fixed anatomical connections in the brain is modulated by the dynamical state of the communicating neural circuits, which allows the quick formation and removal of effective 'communication' links (Friston, 1994; Massimini et al., 2005). The phase difference between the rhythms of two neural groups is a potential marker for the strength of the effective link between them (Womelsdorf et al., 2007; Bosman et al., 2012). This view is particularly appealing in the case of gamma oscillations due to the gamma rhythm's modulation of excitability and firing rate (Fries, 2005).

The behavior illustrated in Fig. 4.1e does not require the local circuit to be a WTA. The WTA can be replaced by any neural circuit as long as this circuit displays a number of distinct firing patterns based on external input and a memory of past firing patterns. Some distinct part of each firing pattern should be characterized by a high enough firing rate so that it can influence the firing pattern in another neural circuit when the two are coupled.

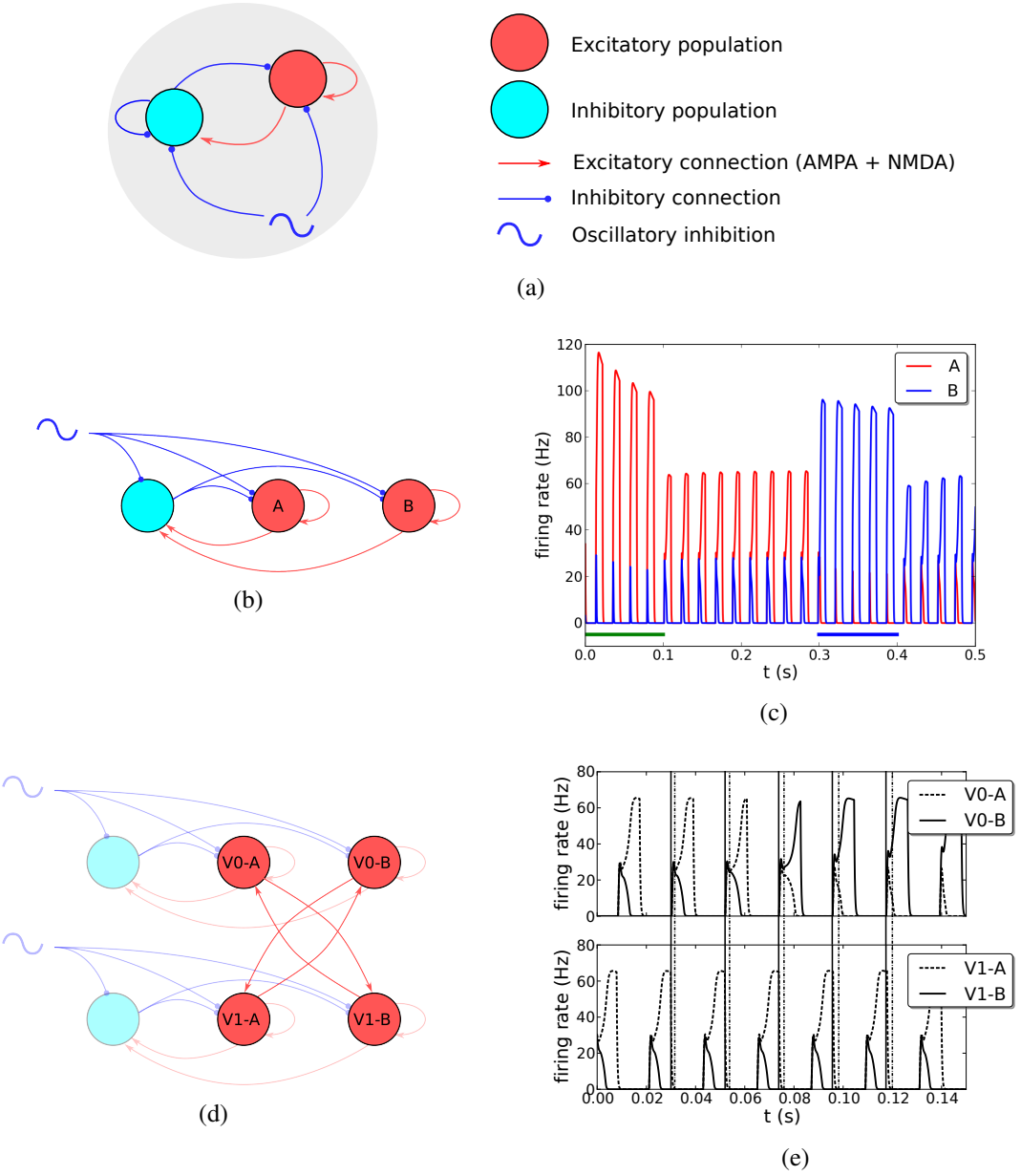


Figure 4.1: (a) Generic structure of a local circuit that undergoes oscillatory inhibition. Oscillatory inhibition is modeled as a periodic rectangular function having an on (high) phase and an off (low) phase. (b) A binary WTA circuit modulated by oscillatory inhibition. (c) Simulation results of the binary WTA. Oscillatory inhibition periodically shuts down all activity. External input (horizontal bars) can bias the winner selection process and in its absence, the identity of the winning excitatory population is maintained across the inhibition cycles. (d) Coupling two WTA circuits, V0 and V1, to enforce the consistency condition V0≠V1. (e) Simulation results of the network in (d) when the frequency of the oscillatory inhibition is slightly different in the two WTA circuits. Solid vertical lines indicate the times when oscillatory inhibition switches off in V0 and dotted vertical lines indicate the times when oscillatory inhibition switches on in V1. The phase difference between the two WTAs is continuously changing and eventually V1 is able to influence V0 to satisfy the consistency condition.

4.3. Properties of the Network Trajectory and Solving Constraint Satisfaction Problems

A WTA circuit can be involved in a number of different consistency conditions. As the rhythmic inhibition wears off in one WTA circuit, other WTA circuits that are coupled to it can influence its state so as to satisfy the consistency conditions encoded in the pattern of coupling connections (see Fig. 4.1d for example). Exactly which consistency condition, if any, gets satisfied by the new state/winner of the WTA circuit depends on the level of activity in the WTA circuits connected to it, which in turn depends on the phase of the inhibition cycle in these WTA circuits. Since the phase relations between the WTA circuits are continuously changing, the relative strengths of the different consistency conditions are also continuously changing.

If we consider each WTA as a discrete-valued variable whose value is the identity of the last winning population, we can enumerate all possible configurations of a network of interacting oscillatory WTA circuits; for example, a network of N coupled WTA circuits with two excitatory populations each can have 2^N possible configurations. Define $d(t)$ as the discrete-valued, continuous-time, dynamical variable that denotes the configuration of the network at time t. This variable is updated each time the local oscillatory inhibition shuts down the activity in a WTA circuit in order to reflect the identity of the WTA circuit's last winning population at that time.

To prove results about the trajectory of $d(t)$, we assume the frequencies of the inhibitory rhythms in different WTAs are incommensurable, i.e. not rational multiples of each other. We also assume that the part of the oscillation cycle in which inhibition is high/active is always longer than the part in which it is low/inactive across all WTAs. Many networks presented in this chapter violate the latter assumption and we observe empirically that the following two results still hold. This latter assumption is, however, needed to prove the following results in the general case.

Proposition 1. *The only fixed points of $d(t)$ are the configurations that satisfy all consistency conditions.*

Proposition 2. *$d(t)$ is not periodic as long as it has not reached a fixed point.*

The proofs are given in appendix D. The network thus traverses the space of possible WTA configurations in an irregular, aperiodic manner until it finds a fully consistent configuration. External inputs can clamp the states of some WTAs, i.e. force particular populations to win the competition in each inhibition cycle. In this case, the rest of the network would search for configurations that satisfy the consistency conditions, given the states of the input-clamped WTAs. We illustrate this behavior using a hard 9×9 Sudoku instance (with one unique solution) that is shown in Fig.4.2a. The network embodying this problem uses a WTA circuit with 9 excitatory populations to represent each square. The index of the winning population in each WTA encodes the number in the underlying square. The Sudoku constraints are implemented by coupling each square/WTA to every other square/WTA in the same row, column, or 3×3 block with a pair-wise connection scheme that encodes an inequality constraint: each excitatory population in one square/WTA is connected to all the excitatory populations in the other square/WTA *except* the excitatory population with the same index (Fig. 4.1d shows an inequality constraint in the binary WTA case). Each WTA will then try to force all the other WTAs in the same row, column, and 3×3 block to encode a different number. External input forces every WTA that represents a pre-filled square to always encode the pre-filled number. For example, the top left WTA/square receives external input so that population 8 always wins in each inhibition cycle.

Figure 4.2b shows the distribution of the time taken by the network to find the unique solution starting from random initial conditions in 100 trials. The average frequency of oscillatory inhibition in each WTA/square is 50 Hz. In all trials, the network manages to find the solution to the Sudoku problem. A distinguishing feature of the proposed network is that the network stabilizes at the solution to the constraint satisfaction problem as predicted by proposition 1. This is in contrast to previous approaches that use stochastic neural networks to search the space of possible solutions and in which the fully consistent solution is transiently visited for a small fraction of time (Habenschuss et al., 2013). The network does not get stuck at non-optimum configurations, as opposed to architectures based on Hopfield networks that tend to get stuck at locally optimal solutions (Hopfield, 2008). The novel search dynamics employed by the proposed networks are driven by local rhythmic inhibition which continuously and irregularly modulates the effective interaction strength between the WTAs. This causes the set of active constraints or consistency conditions to continuously change, which allows the network to escape from locally optimal configurations. Only the globally optimal configuration is stable because, irrespective of which consistency conditions are active, these active consistency conditions are fulfilled and so will act to preserve the current configuration.

As shown in Fig. 4.2c, the network trajectory is far from being a brute-force exploration of all the possible $\sim 10^{52}$ configurations of the unfilled squares. The network quickly converges to configurations that violate a small fraction of the 810 inequality constraints. There is thus a strong 'greedy' element in the network behavior. However, the network is still able to escape from highly consistent local optima and visit less consistent configurations on its way to the globally optimal solution. The path to the global solution, however, still predominantly passes through highly consistent states. As shown in Fig. 4.2d, the network quickly yields reasonably good solutions that only violate a few constraints, and on average tends to occupy better configurations the longer it is allowed to run until it finally finds and stabilizes at the globally optimal solution. Even though the proposed networks cannot beat state of the art algorithms for solving constraint satisfaction problems, they highlight a form of distributed, non-stochastic, biologically realistic dynamics that can execute a search for optimal solutions in a complex configuration space.

4.4. Sampling Analogy

According to proposition 1, if the consistency conditions encoded in the coupling connections can not all be simultaneously satisfied, the network will never settle in one configuration. This is the case for the network shown in Fig. 4.3. Figure 4.3b shows that the network tends to spend more time in configurations that violate fewer consistency conditions. Figure 4.3c shows the relative duration of time the network spends in each configuration. The plot in Fig. 4.3c can be interpreted as the plot of a probability distribution over the possible configurations of the network. We take this stochastic interpretation further by approximating the network dynamics by a Markov Chain Monte Carlo (MCMC) sampling process. Based on the network topology, we construct a stochastic Markovian transition operator T that approximates the network trajectory (The construction of the MCMC operator is described in appendix E).

We compare the normalized duration the trajectory of $d(t)$ spends in a particular configuration to the normalized frequency of occurrence of this particular configuration in the sequence of samples generated by the stochastic Markovian transition operator T. The results are shown in Fig. 4.3c for the example network. The two sets of statistics are similar even though they were generated by two widely different classes of systems: the network, which is deterministic, continuous-valued and running in continuous time, and the MCMC sampler, which is stochastic,

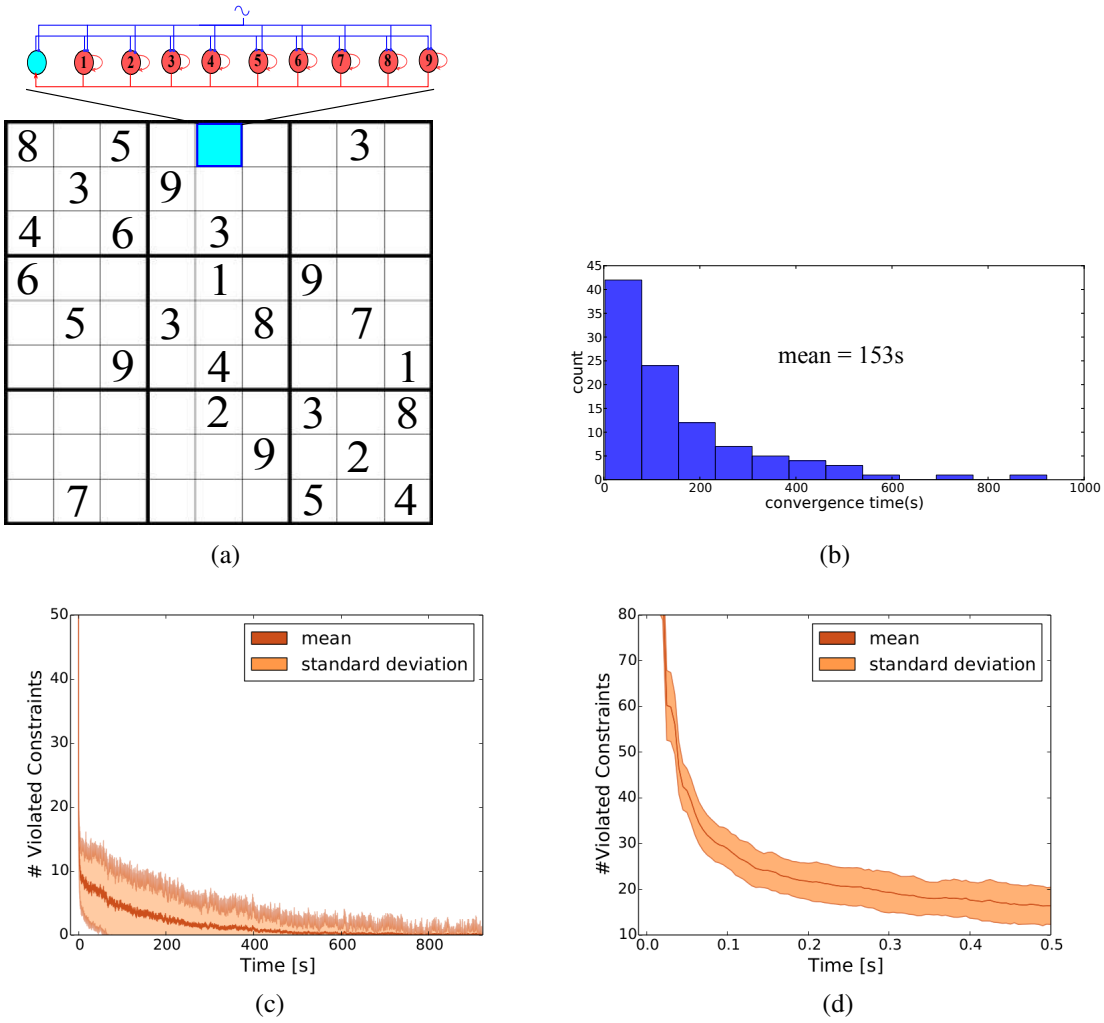


Figure 4.2: (a) A 9×9 hard Sudoku instance. Each square is represented by one WTA with 9 excitatory populations, which receive oscillatory inhibition. (b) Histogram of the convergence time to the unique solution in 100 trials starting from different initial conditions. (c) Mean and standard deviation of the number of inequality constraints violated by the Sudoku network at each time point across the 100 trials. (d) A zoom-in of the first 0.5 s in (c).

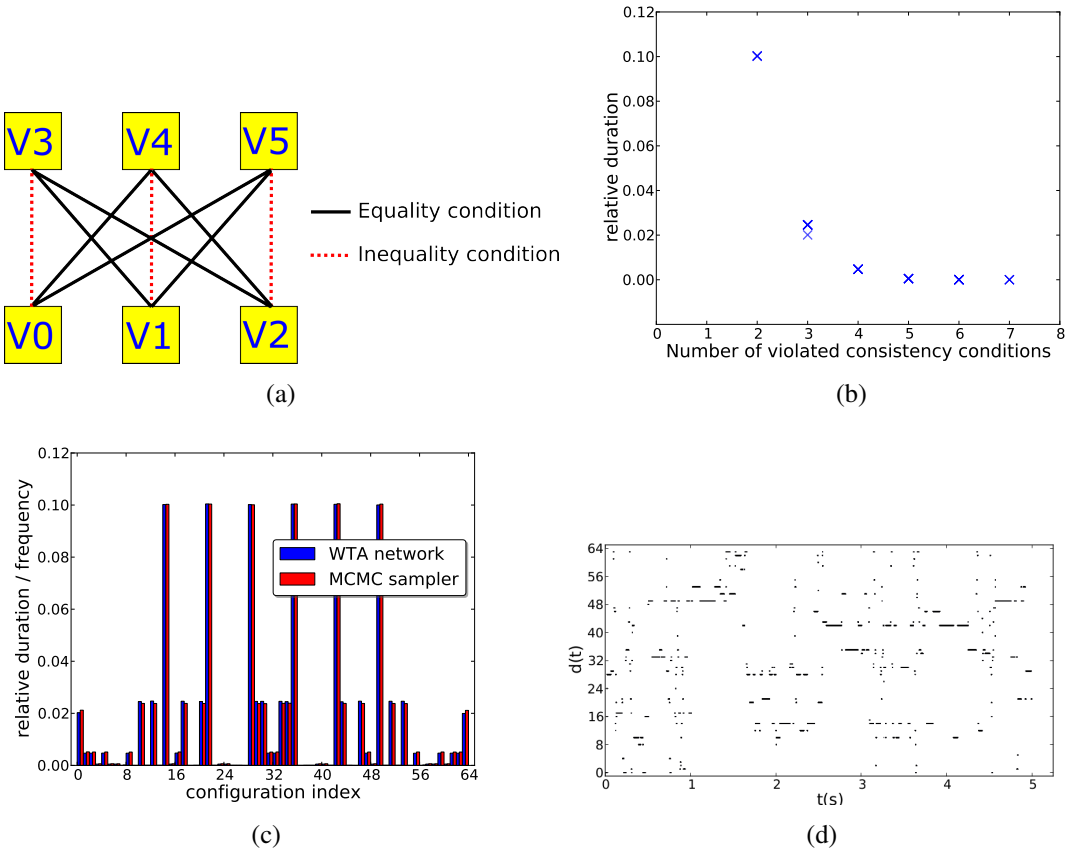


Figure 4.3: (a) Topology of a network with irreconcilable consistency conditions. Each square corresponds to a binary WTA. (b) Normalized duration of time spent by the network in each of the 64 configurations as a function of the number of violated consistency conditions. Each cross denotes one configuration. (c) Normalized durations of the 64 network configurations obtained by simulating the actual network for 1×10^6 seconds and the relative frequencies of occurrence of these configurations in a sequence of 10^8 samples generated by a stochastic MCMC operator constructed to approximate the network trajectory. The configuration index is the decimal value of the binary string encoding the states of V0 to V5. (d) Trajectory of the dynamical variable $d(t)$ for the network in (a) in the first 5 seconds.

discrete-valued and running in discrete steps. The sampling analogy is reinforced by the aperiodicity of $d(t)$ which yields an irregular trajectory of the network state that is akin to a truly stochastic trajectory as shown in Fig. 4.3d. The difference between two probability distributions p and q can be quantified using Kullback-Leibler (KL) divergence:

$$KL(p,q) = \sum_{x_i} p(x = x_i) log \left(\frac{p(x = x_i)}{q(x = x_i)} \right)$$

For the network shown in Fig. 4.3, $KL(p_{network}, p_{MCMC}) = 9.0 \times 10^{-4}$nat. In appendix E, we show that the stochastic approximation is still good for various network sizes and inter-WTA connection densities and analyze the properties of the MCMC operator as well as discuss in more detail the assumptions underlying the stochastic approximation. Stochastic, sampling-based inference has often been invoked as a theoretical model for how the brain is able to represent and manipulate complex, multi-modal probability distributions (Gershman et al., 2012; Sundareswara & Schrater, 2008; Buesing et al., 2011). The results presented here provide a novel neural framework that reproduces MCMC sampling dynamics and relates these dynamics to local rhythmic inhibition, rather than to noise processes in the neuron or synapse models.

Of course, given the statistics of the network trajectory, it is trivial to construct an MCMC operator that has these statistics as a stationary distribution using any of the standard sampling methods such as Gibbs sampling. The MCMC operator described in this section, however, was constructed *a priori*, i.e. before simulating the network, as an approximation of the network trajectory.

4.5. Effect of Noise and non-zero Coherence

The deterministic networks we present exploit the continuously shifting phase relations between different WTAs in order to explore the space of possible network configurations. The exploration process is not driven by noise. The addition of noise sources in the network, however, can modulate this exploration process. We added white Gaussian noise to the input of all LTUs and random fluctuations to the phases of all inhibitory rhythms (see appendix C for more details). Figure 4.4a shows that noise 'smears out' the distribution of relative durations by biasing the trajectory towards low-duration states and away from high-duration states. This smearing out effect is more pronounced for larger random fluctuations. This is reminiscent of how higher noise/temperature in a thermodynamical system acts to increase the system entropy by making the probability distribution over the system states more uniform. There is a crucial difference between these networks and thermodynamical systems, however: at the zero noise (zero temperature) level, a thermodynamical system quickly freezes at a single state while our networks will continue to explore the configuration space.

Gamma rhythms in distant neural assemblies often exhibit non-zero but still imperfect coherence (Bosman et al., 2012), even if the local Gamma frequency changes in a time and stimulus-dependent manner (Roberts et al., 2013). We model this phenomenon by coupling the phases of the inhibitory rhythms in different WTAs according to the Kuramoto model (Strogatz, 2000) in order to realize phase relations that are consistently near zero. This phase coupling was introduced between variables/WTAs V0 and V3 in the network in Fig. 4.3a and all network components were perturbed by random fluctuations. Figure 4.4d shows that V0-V3 coherence increases the duration of all configurations in which the V0-V3 consistency condition is satisfied and decreases the duration of all configurations in which it is not. Increased coherence between two WTAs thus leads to stronger interaction and makes it more difficult to violate the consistency conditions between them. The increase in interaction strength due to increased coherence

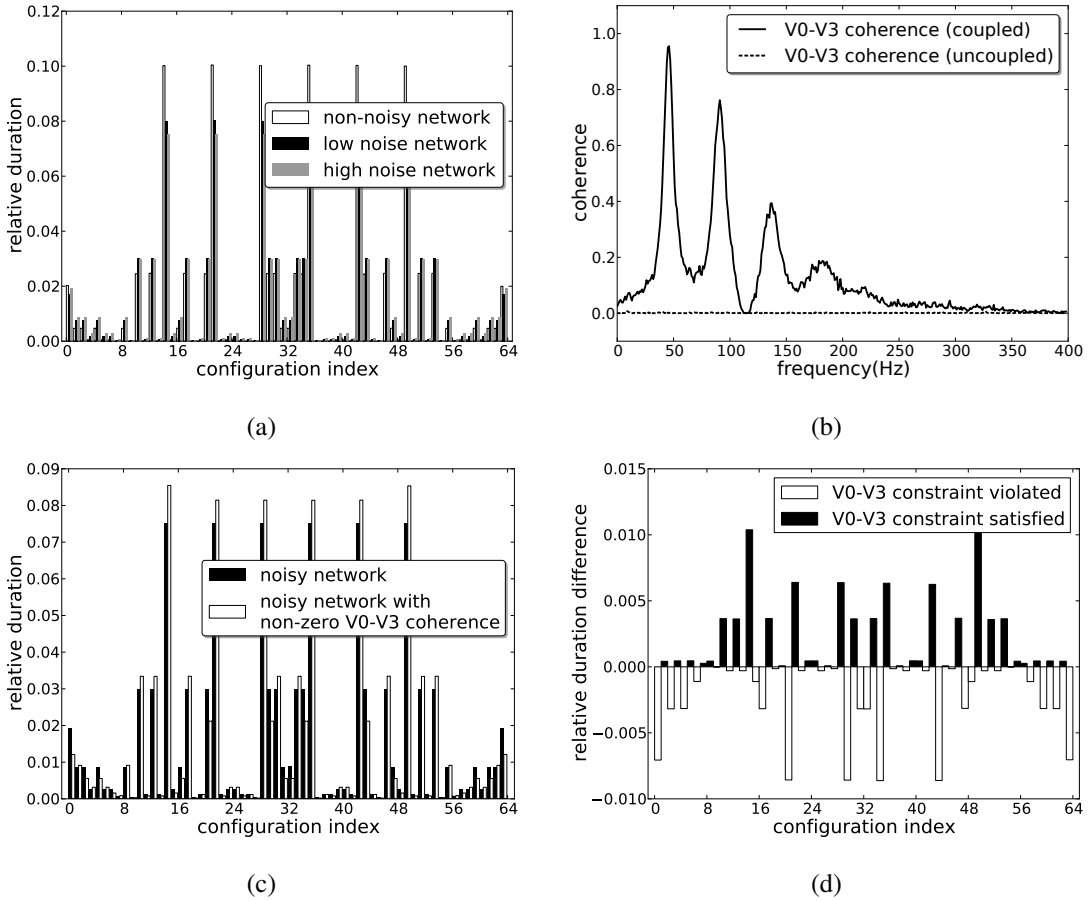


Figure 4.4: (a) Statistics of the trajectory of the network in Fig. 4.3a in the non-noisy, low noise, and high noise cases. (b) Mean square coherence between the oscillatory inhibition in variables V0 and V3 in the high noise case when their phases are coupled and when they are uncoupled. (c) statistics in the high noise case when V0 and V3 are phase coupled and when they are uncoupled. (d) Change in the relative (normalized) duration of each network configuration when V0-V3 become phase coupled. Data was obtained from (c). Black bars indicate configurations in which the V0-V3 consistency condition is satisfied while white bars indicate configurations in which the consistency condition is violated.

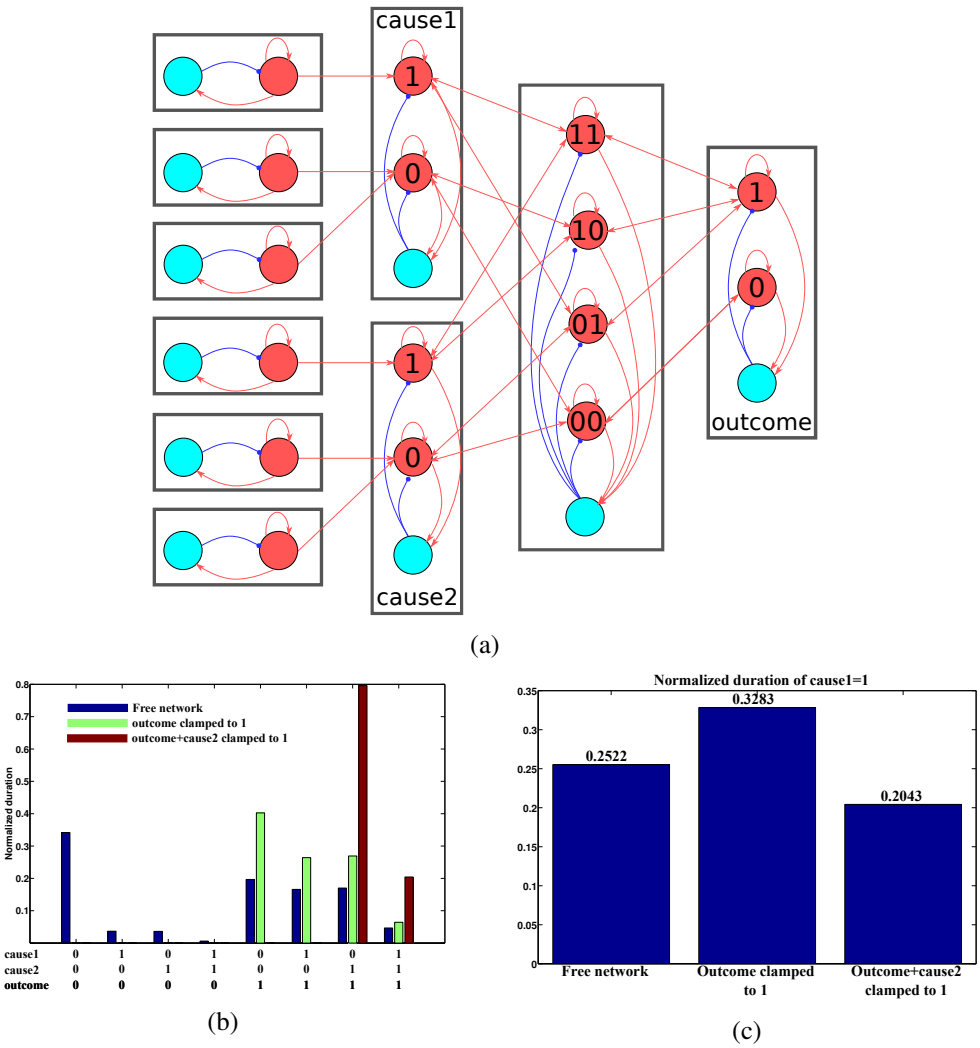


Figure 4.5: A High-order consistency condition gives rise to the explaining away effect. (a) Network topology. The oscillatory inhibition input to each WTA is not shown. The high-order consistency condition encoded in the network connectivity requires the outcome to be an OR function of the two causes. (b) Normalized duration of occurrence of the eight possible states of the outcome, cause1, and cause2 WTA circuits under three conditions: network running freely; outcome WTA clamped to 1; and outcome and cause2 clamped to 1. In each condition, the network was simulated for 10^5 seconds. (c) Normalized duration of the configurations in which cause1 is 1 under the three aforementioned conditions. We interpret this as the marginal probability of cause1 = 1 under the three conditions.

between neural assemblies is consistent with many pieces of experimental evidence (Bosman et al., 2012; Fries, 2005).

4.6. High-order Consistency Conditions and the Explaining Away Effect

High-order consistency conditions, i.e. consistency conditions involving more than two variables, can capture complex dependencies between the states of multiple WTA circuits that can not be captured by pairwise connections between the WTA circuits. One example is the dependencies that give rise to the 'explaining away' phenomenon which is believed to underlie several aspects of visual perception (Knill & Kersten, 1991; Kersten & Yuille, 2003). Explaining away occurs in situations in which many causes can give rise to a particular outcome. When this particular outcome is observed, the individual likelihood of each cause increases as at least one of the causes is needed to explain the outcome. If in addition, one of the causes is observed to have taken place, the likelihood of the other causes goes down again. The observed cause explains away the other causes as they are no longer needed to justify the outcome. This form of intercausal reasoning establishes a dependency between the different causes of a particular outcome when this outcome is observed. Fig. 4.5a shows a network used to model the dependencies that give rise to the explaining away effect. Two causes, cause1 and cause2, can each give rise to an outcome. The causes and the outcome are binary and they are represented by one WTA each. The causes and the outcome are coupled to a hub WTA so that the binary outcome is effectively an OR function of the two causes. For each cause, we introduce three unitary WTA circuits where two unitary WTA circuits project to the 0 population and one projects to the 1 population of the cause WTA. That in effect introduces a 'prior' over the states of the two causes that assigns a higher likelihood to them being 0. Each WTA oscillates at a slightly different frequency. The network as a whole can not have a consistent state as each cause can not simultaneously satisfy the contradictory requirements of the unidirectional consistency conditions coming from the unitary WTA circuits.

Figure 4.5b shows the durations the network spends in the eight possible configurations of the two causes and the outcome. In the free-running network, there are four configurations that satisfy the high-order consistency condition (the OR relation). Due to the action of the 'prior', configurations that satisfy the high-order consistency condition have a higher 'probability' if more of the causes are 0. The strong 'prior' can override the high-order consistency condition and assign a high 'probability' to a configuration in which the two causes are 0 and which does not satisfy the high-order consistency condition. Similar trends can be seen when the outcome WTA is clamped to 1 and when both the outcome WTA and the cause2 WTA are clamped to 1. Figure 4.5c highlights the explaining away effect. In the free running network, the normalized duration for which cause1 is equal to 1 is 0.2522. This can be interpreted as the 'marginal probability' of cause1 = 1. When the outcome is clamped to 1, this marginal probability increases as one of the causes is needed to explain the outcome (it can now be interpreted as the marginal probability conditioned on outcome = 1). When in addition cause2 is clamped to 1, the probability of cause1 = 1 goes down again as cause1 is not needed to explain the outcome when cause2 is 1.

4.7. Learning the Consistency Conditions

The plasticity rule in Eq. 4.2 was developed by Lorenz Muller.

Like any connectionist architecture, the knowledge in the networks we describe is encoded in the coupling connections between the WTA circuits. These coupling connections represent the consistency conditions. By introducing Hebbian-like plasticity in the inter-WTA coupling

connections, the strengths of these connections will be continuously modulated by the network configuration. Hence, external input that forces particular configurations on the network can reorganize the pattern of coupling connections. We show in the next section that this reorganization occurs in such a way so as to maximize the consistency of the input-imposed configurations as interpreted according to the number of consistency conditions these configurations violate. This will in turn lead to a higher occurrence of the input-imposed configurations when the network is running freely.

A central question in such a learning scheme is how the plasticity rule can distinguish between configurations that are input-imposed, and thus should be learned, and configurations that arise naturally when the network is running freely. One possible solution to this problem is to have the input impose a particular configuration for a long time and thus distinguish this configuration from the others by virtue of its longer persistence. This, however, leads to slow learning and there is no guarantee that a configuration generated in the free-running network will not persist just as long. This central question also arises in stochastic connectionist architectures that represent a probability distribution by sampling, and that learn the probability distribution by example, such as restricted Boltzmann machines (RBM). The learning rule used in RBMs (Ackley et al., 1985; Hinton, 2002) assumes the network has access to a signal indicating whether a configuration is input-imposed or not. This signal switches the plasticity rule modulating the weights in the network between Hebbian and anti-Hebbian modes. It is not clear what the biological analogue of such a scheme could be.

To distinguish an input-imposed configuration as a configuration that should be learned, i.e. that should serve as a model for consistent configurations, we have external input synchronize the activity of the WTA circuits it targets so that the oscillatory inhibition in these WTA circuits has a common frequency and a zero phase difference. In this way, the full or partial configurations imposed by the input have the distinguishing dynamical feature of synchronized WTA circuits. In our networks we use synchronization as a dynamical marker for configurations that should be learned. We do not model how the input is able to synchronize the WTA circuits; we simply enforce synchronization among WTA circuits receiving external input directly in our simulation environment.

We now introduce a plasticity rule that acts on the weights of the inter-WTA coupling connections:

$$\begin{aligned} \frac{dw(t)}{dt} = d(w) \quad &+ \quad \eta_{up} \cdot [r_{pre}(t) - \theta]^+ \cdot [r_{post}(t) - \theta]^+ \\ &- \quad \eta_{down} \cdot [r_{pre}(t) - \theta]^+ \cdot [\theta - r_{post}(t)]^+ \end{aligned} \tag{4.2}$$

where

$$\begin{aligned} d(w) &= \begin{cases} d_{up} & \text{if } w(t) > w_{mid} \\ -d_{down} & \text{otherwise} \end{cases} \\ [x]^+ &= \max(0, x) \end{aligned}$$

$w(t)$ is the weight. $r_{pre}(t)$ and $r_{post}(t)$ are the firing rates of the source (presynaptic) and target (postsynaptic) populations respectively. This rule is a variation of the Bienenstock-Cooper-Munro (BCM) rule (Bienenstock et al., 1982) with hard weight bounds w_{min} and w_{max} (not shown in the equation) and the requirement that $r_{pre}(t)$ has to exceed a threshold, θ, in order to induce any change in the weight $w(t)$. If this requirement is met, potentiation is induced if the postsynaptic activity, $r_{post}(t)$, is above the threshold θ, and depression is induced if the postsynaptic activity is below the threshold. The rates of potentiation and depression induction are controlled by η_{up} and η_{down} respectively. The rule captures the way potentiation and depression

induction depend on the pre- and post-synaptic firing rates (Sjöström et al., 2001). The rule contains a second component that slowly forces the connection weight to either w_{min} or w_{max} depending on whether the weight is below or above $w_{mid} = \frac{1}{2}(w_{max} + w_{min})$ respectively. The rule is thus bistable. The strength of the bistability drift is controlled by d_{up} and d_{down}.

In a rate based network, we are unable to capture the way gamma synchronization enhances plasticity by forcing neurons to spike within a narrow time window (see for example (Lee et al., 2009)). However, the plasticity rule in Eq. 4.2 results in the same functional effect of enhanced plasticity in the inter-WTA connections when the WTA circuits are synchronized. In order for a depressed connection to potentiate, both pre- and postsynaptic rates need to be large at the same time for many consecutive cycles in order to overcome the bistability drift. This will only reliably happen if the oscillatory inhibition in the pre- and post-synaptic WTA circuits have a phase difference that is around zero for many cycles so that the peaks of excitatory activity in the WTA circuits coincide.

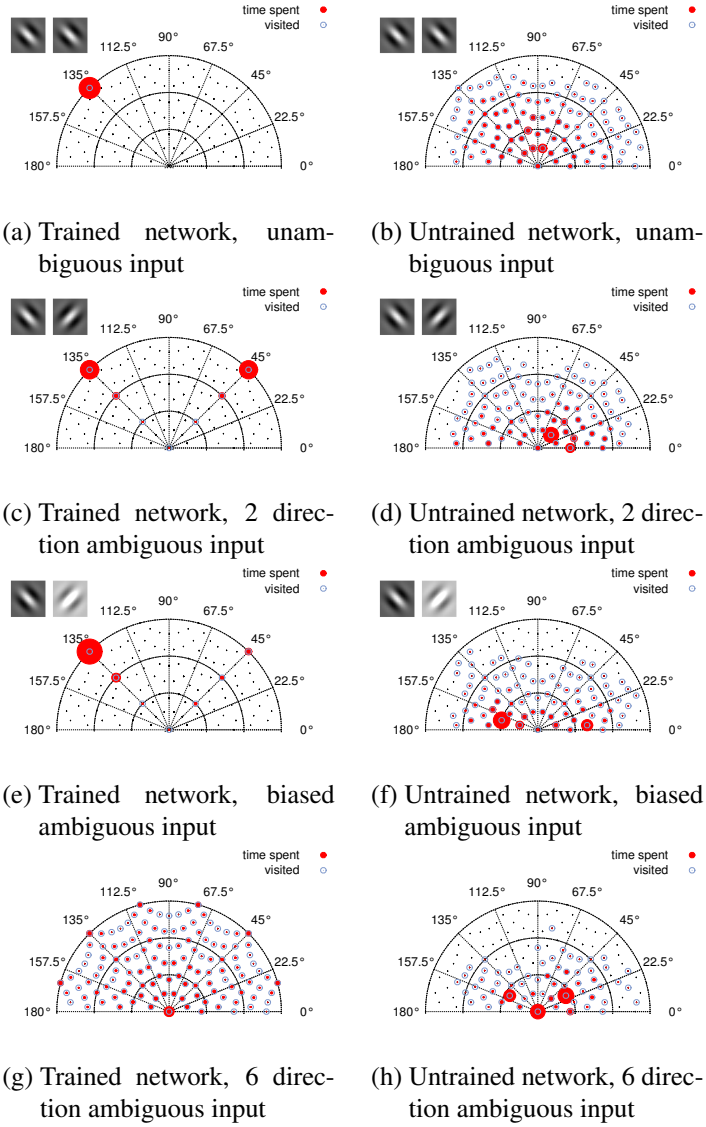
4.8. Perceptual Multi-stability

Lorenz Muller has exploited the sampling like-behavior of the presented neworks in the absence of fully consistent states to model perceptual multi-stability phenomena where a subject's perception switches in a seemigly stochastic fashion between two maximally (but not fully) consistent percepts. Details of the network model are given in Mostafa et al. (2015c). The particular form of perceptual multi-stability considered is binocular rivalry, which has been well studied experimentally (Mamassian & Goutcher, 2005) as well as in theory (Gershman et al., 2012). In binocular rivalry experiments, each eye is shown a differently oriented grating. The subject's perception then continuously switches between the two orientations (Mamassian & Goutcher, 2005). The results presented in this section are primarily the work of Lorenz Muller and are presented in more detail in Mostafa et al. (2015c).

The simulations run have two distinct phases, a training phase, in which the network receives consistent input (corresponding to the two eyes seeing gratings with the same orientation), and a testing phase, in which we clamp a subset of the WTAs at various orientation patterns and decode the activity of the rest of the WTA circuits to obtain the orientation 'perceived' by the network. During the two phases we use *the same synaptic rule and parameters*; learning is effectively enabled/disabled by providing/withholding a synchronizing oscillatory input to the inhibitory oscillators of all WTAs.

Figure 4.6 shows the distribution of times that the network spends at each orientation vector. The possible orientation vectors that the network can encode are shown as black dots, and the time spent encoding each orientation vector is represented by the size of the grey circles surrounding the black dots. The angle of the orientation vector encodes the orientation 'perceived' by the network while the vector length encodes the network confidence in that percept. As shown in Fig. 4.6a, the trained network represents an orientation with high confidence if that orientation is unambiguously presented to the network. The network alternates between encoding two different orientations when it receives input that clamps different parts of the network to different orientations as shown in Fig. 4.6c. In an ambiguous input setting where one orientation is presented more strongly than another, the network spends more time encoding the more strongly presented orientation as shown in Fig. 4.6e. The untrained network has not learnt an internal consistency model that it can use to interpret the input. It thus fails to make sense of even unambiguous inputs and spends most of its time in low confidence states.

In the ambiguous input case where two different and equally strong orientations are presented (Fig. 4.6a), the switching times between the two percepts in the network closely match human



(a) Trained network, unambiguous input

(b) Untrained network, unambiguous input

(c) Trained network, 2 direction ambiguous input

(d) Untrained network, 2 direction ambiguous input

(e) Trained network, biased ambiguous input

(f) Untrained network, biased ambiguous input

(g) Trained network, 6 direction ambiguous input

(h) Untrained network, 6 direction ambiguous input

Figure 4.6: The 'probability distribution' of the orientation vector decoded from the WTAs in the network under different conditions. Black dots indicate states that can be encoded by the network, and black empty circles indicate visited states. The area of the filled grey circle at each visited location is proportional to the time spent in that state (scale differs between plots). The inset Gabor patches indicate the stimuli presented to the left and right eye in analogous experimental settings. (a) A network trained on consistent inputs propagates the clamped angle of one input WTA to the other WTA circuits and stably represents that angle. (b, d, f, h) An untrained network spends most of the time in 'low confidence states' where the WTA circuits encode different angles in all input cases. (c) The trained network switches between two interpretations of an ambiguous input. (e) Increasing the sterngth of the $135°$ input makes that interpretation more probable compared to the $45°$ interpretation. (g) The trained network under fully conflicting input preferentially visits consistent states, where all populations represent the same direction. 'Low confidence' states are also visited.

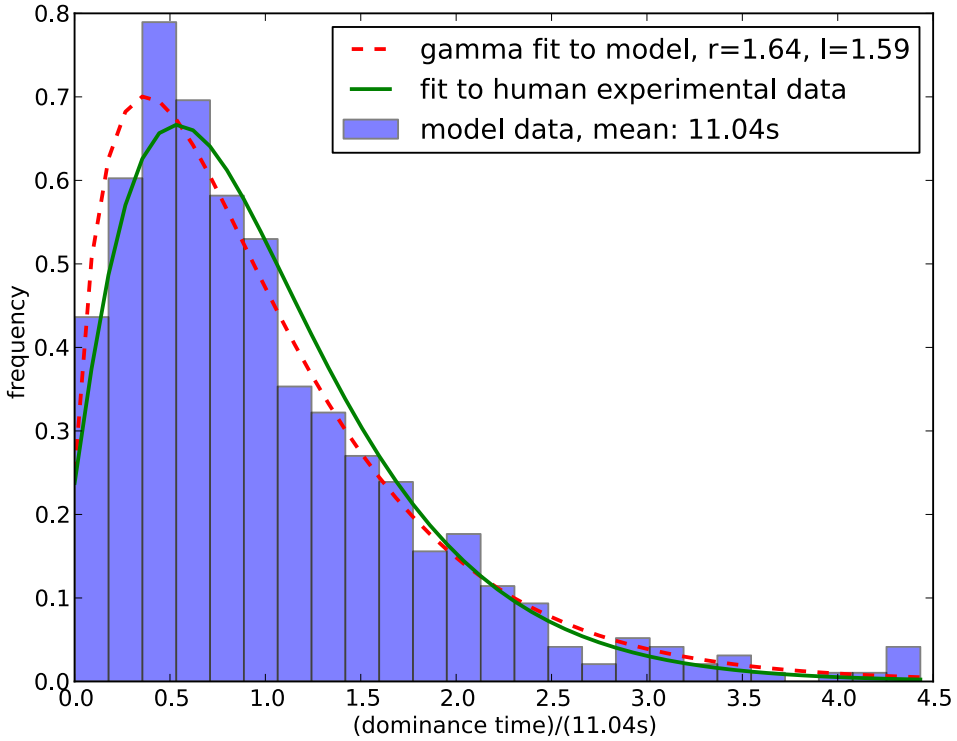


Figure 4.7: The perceptual switching times produced by our network model fit a gamma distribution that largely agrees with experimental data from Mamassian & Goutcher (2005).

experimental data (Mamassian & Goutcher, 2005), see Fig.4.7. The histogram is approximately a gamma-distribution. The mean time between switches is around 11 seconds, with a maximum of 49 seconds.

4.9. Discussion and Testable Predictions

Several theories postulate that the key computational machinery of sensory and association cortices is devoted to solving best-match problems (Minsky & Papert, 1969; Rumelhart & McClelland, 1986) where a number of interacting areas try to agree on a global interpretation that best explains sensory data while being consistent with an internal model of the environment, or prior (Berkes et al., 2011; Friston, 2003). We showed that oscillatory inhibition that gives rise to rhythmic modulation of firing rate and sensitivity to external inputs in the local circuitry allows a neural network to solve best-match problems. If no fully consistent network configuration exists, the network trajectory wanders aperiodically and can be approximated by a stochastic sampling process. We have shown that non-stochastic networks can reproduce effects that have commonly been modeled using stochastic networks such as perceptual multi-stability and the explaining away effect. Our results show that physical noise mechanisms are not strictly necessary for networks to explore complex configuration spaces without becoming trapped in locally optimal configurations.

The search for consistent configurations in the networks described has both a greedy and an

exploratory aspect. The latter aspect is essential for escaping locally optimal configurations. A critical parameter that makes the network either more exploratory or more greedy is the fraction of each cycle in which oscillatory inhibition is on (high). If oscillatory inhibition is on for most of the cycle, the WTAs are inactive for longer periods of time and the consistency conditions are thus inactive (not enforced) for most of the time. This leads to a more exploratory search as it is more probable that only few consistency conditions are active at any one time, causing the network to move to configurations that might be inconsistent with the majority of consistency conditions. If, on the other hand, oscillatory inhibition is off for most of the cycle, more WTAs and more consistency conditions are active at any one time (on average), and the network exhibits a more greedy behavior as it tries to conform to the influence of a large number of consistency conditions.

The behavior of the described networks remains qualitatively unchanged for a wide parameter range. For the single WTA parameters, activity should be self-sustaining at a finite rate. As for the inter-WTA connections, they should be strong enough to enable the activity in the winning population in one WTA to influence the winner selection in a connected WTA during the winner selection phase and counteract the hysteresis mechanism mediated by the intra-WTA recurrent NMDA currents as well as the inter-WTA NMDA currents that persist from previous oscillation cycles. These conditions are satisfied by a fairly wide parameter range (Rutishauser et al., 2012). The fundamental network behavior of searching for maximally consistent states does not require any particular distribution of oscillatory inhibition frequencies, provided the frequencies are incommensurable. The distribution of inhibition frequencies, however, impacts how fast the network is able to escape from locally optimal configurations. Wider frequency distributions cause the phase relations to change more quickly, which would lead to faster changes in effective connectivity and faster switching between the different configurations. The frequency distribution was chosen in the Sudoku example to optimize time to convergence and in the binocular rivalry experiment to more accurately match experimental data. Tuning the rate at which phase relations change could alternatively be achieved through dynamic modulation of coherence between the neural assemblies/WTAs.

There is a natural relationship between the WTA mechanism and the Pyramidal Interneuron Gamma (PING) (Tiesinga & Sejnowski, 2009; Whittington et al., 2011) rhythm which is generated through the reciprocal interaction between pyramidal and inhibitory interneurons. As inhibition ramps down in the PING network, the excitatory neurons race to reach the spiking threshold and the neuron(s) receiving the strongest input will be the first to overcome the waning inhibition and spike. They will then excite the inhibitory interneurons, which will fire, raising the level of inhibition in the network and shutting down the excitatory neurons. Inhibitory postsynaptic currents eventually die down and the cycle repeats (de Almeida et al., 2009; Fries et al., 2007). The WTA mechanism we use is quite different as it is rate-based, makes use of direct competitive dynamics through a common inhibitory population, and does not model the PING Gamma generation mechanism. The results obtained, however, are applicable to the spiking PING networks with the race-like WTA mechanism. Connected PING networks with different natural frequencies interact when they have the correct phase relations. At the correct phase relation, spikes from the winning pyramidal neurons in one PING network will affect the winner selection in a connected PING network. Similar to our networks, this would allow connected PING networks to explore different configurations in search of firing patterns that are in line with the network connectivity.

Since the activity of one PING network can impact the activity as well as the oscillation phase of connected networks, interconnected PING networks may exhibit synchronization patterns which would act to modulate the effective strength of the connections between them. This is similar to how non-zero coherence affects the strengths of the consistency conditions in our

network (see Fig. 4.4d). Since we do not model the Gamma generation mechanism, we can only use an abstract phase coupling scheme to achieve non-zero coherence. This artificial scheme has limited biological relevance and will not yield natural coherence profiles.

Increased Gamma-band coherence between distant neural groups has been found to increase the strength of their mutual influence (Bosman et al., 2012; Fries, 2005) making the coherence level of the gamma oscillations a candidate mechanism for quickly modulating the effective strength of anatomical connections (the 'communication through coherence' hypothesis). In agreement with this effect, we have shown that if oscillatory inhibition in the WTA circuits were forced to exhibit some coherence, which leads to consistent and favorable phase relations (phase differences that are around zero), communication between the coherent WTA circuits would be more effective as the consistency conditions inside the coherent set would be more difficult to violate (see Fig. 4.4d).

We have used synchronization as a way to trigger learning. There is evidence that long-range synchronization is required for effective learning and memory formation (Weiss & Rappelsberger, 2000; Fell & Axmacher, 2011) and that it is a candidate mechanism for establishing associations between disparate neural groups (Miltner et al., 1999). Gamma band synchronization in particular is well-suited for establishing associations through STDP mechanisms (Axmacher et al., 2006) as neurons fire preferentially at a particular phase of the gamma cycle (when the oscillatory inhibition is off), thereby aligning their spikes to a precision of a few milliseconds, which is a suitable window for inducing synaptic potentiation or depression (Markram et al., 1997).

Our perceptual multistability model is, to our knowledge, the first model that reproduces perceptual multistability phenomena without making use of explicit stochastic dynamics. An experiment that could provide evidence as to whether our model does underlie multi-stable perception would investigate the relationship between perceptual switching times and how fast Gamma-band phase differences between fields measured at different points on the visual cortex change. Faster changes in phase relations translates to a more exploratory behavior in our model as the effective network connectivity changes more rapidly and should translate to faster switching times between the different percepts.

This chapter concludes the first part of this thesis. In the second part, I address issues related to developing event-based neuromorphic systems. In particular, I describe in chapter 7 how the dynamics of the oscillatory networks described in this chapter can be simplified and used to engineer efficient hardware systems for solving constraint satisfaction problems. In chapter 8, I describe how these oscillatory dynamics can be used to approximate the behavior of artificial stochastic neural networks such as restricted Boltzmann machines.

Part II.

Neurally-inspired Physical Architectures

5

Automated Synthesis of Address Event Representation (AER) Interfaces

---

This chapter is based on the paper "Automated synthesis of asynchronous event-based interfaces for neuromorphic systems" by Hesham Mostafa, Federico Corradi, Marc Osswald, and Giacomo Indiveri, European Conference on Circuit Theory and Design (ECCTD), Dresden, Germany, 2013

In the second part of the thesis, I investigate topics related to the physical implementation of neurally-inspired computing systems on hardware architectures. In this chapter, I begin by addressing the general problem of developing AER interfaces in an automated fashion. AER interfaces are a crucial component of spike-based neuromorphic systems and it is important to have a robust, flexible, and automated method for generating these interface circuit to speed up the development cycle of neuromorphic chips. In the next chapter, I address another general problem: how can nano-scale memristive devices be used as plastic synaptic elements in a hybrid CMOS-memristor neuromorphic system. I present proof-of-concept physical measurements of a a direct integration between memristive devices and a CMOS neuromorphic chip that implements the spike-based perceptron learning rule from ref. (Brader et al., 2007).

Chapters 7 and 8 pick up again the main thread of this thesis which is the development of novel biologically-inspired models of computation and the efficient implementation of these models on VLSI systems. In chapter 7, I describe how the dynamics of the biologically-inspired oscillatory networks presented in chapters 3 and 4 could be adapted to allow efficient implementation on VLSI chips. The resulting architecture, the *inferenceEngine* architecture, turns out to be quite efficient in solving a variety of hard CSPs. In chapter 8, I investigate how the *inferenceEngine* architecture can be used to implement stochastic networks from the field of machine learning such as restricted Boltzmann machines (RBMs). RBMs are one of the possible building blocks for deep belief networks which have emerged as powerful network architectures in various classification and encoding tasks (Hinton & Salakhutdinov, 2006; Hinton et al., 2006). The results presented in chapter 8 illustrate a potential route for implementing these widely applicable and powerful networks efficiently on VLSI systems.

I now present an automated design approach that leverages the commonly available digital design tools in order to rapidly synthesize asynchronous event-based interface circuits from behavioral Hardware Description Language (HDL) code. As part of the proposed design approach, I describe a verification methodology that is able to reveal early in the design process potential timing failures in the generated circuits. The proposed automated approach was used to synthesize the AER interface circuits in the hybrid CMOS-memristive system described in the next chapter, the prototype chip implementing the *inferenceEngine* architecture described in

chapter 7, as well as multiple generations of multi-neuron neuromorphic chips (Mostafa et al., 2014; Qiao et al., 2015). The AER interface circuits functioned correctly in all these chips.

Neuromorphic event-based systems generally consist of multiple custom hybrid analog/digital VLSI modules that communicate among each other using the AER protocol (Boahen, 2000). In this representation each source or sender node (e.g., a silicon neuron) is assigned an address, and when it produces an event (e.g., a spike) its address is instantaneously put on a digital bus, using asynchronous logic. Destination nodes (e.g., synapses) can decode and "consume" these asynchronous address-events at the time in which they receive them. In the case of single-sender/single-receiver communication, a handshaking mechanism ensures that all events generated at the sender side arrive at the receiver. Typically signals are encoded using a bundled data representation, in which the address of the sending element is conveyed as a parallel word of sufficient length, and two additional lines are required for the handshaking control signals. Systems containing more than two AER modules are constructed by implementing additional purpose arbitration schemes (Fasnacht & Indiveri, 2011; Chicca et al., 2007; Gomez-Rodriguez et al., 2006). In AER systems therefore time represents itself, input and output address-events are transmitted using asynchronous digital pulses that encode the address of the sending node, and analog information is carried in the temporal structure of the inter-pulse intervals and in their mean frequency. If multiple senders generate events simultaneously, an arbitration scheme makes sure that the addresses do not collide, but are transmitted on the bus in sequence, as shown in Fig. 5.1. The arbiters that manage event-collisions are implemented using asynchronous digital logic circuits. These asynchronous digital circuits change the state of their memory elements in response to transitions on the data lines. This is very different from what happens in *synchronous* logic, where all state transitions occur at the edges of a global clock. Synchronous logic is more common than asynchronous one by far. Indeed the predominance of synchronous logic has led to powerful Electronic Design Automation (EDA) tools that greatly accelerate the design process. While there have been recent advances in the research and design of asynchronous logic, especially for neuromorphic systems (Imam et al., 2012; Jin et al., 2010; Merolla et al., 2007), there is still no mature and readily available digital design flow for the development and automated design of asynchronous circuits. As a consequence, most asynchronous digital circuits designed by the neuromorphic engineering community are done manually, following a time consuming and error-prone process. This chapter describes how the widely available synchronous digital design tools can be adapted to automatically synthesize asynchronous AER arbitration and interfacing circuits.

5.1. Automatic Synthesis of Basic Asynchronous Circuits

The automated design flow is shown schematically in Fig. 5.2. It is based on the design flow for synchronous digital circuits. The flow starts by creating a Very high speed integrated circuit Hardware Description Language (VHDL) behavioral model for the required circuit. Standard logic synthesis tools create a gate level netlist using standard logic gates, latches, and flip flops from the behavioral description. A physical layout is then created from the gate level netlist by place and route tools that have access to the layout cells of the standard logic gates. The gate level netlist can be expanded to a transistor level netlist, and annotated with the parasitic elements extracted from the layout. I show how this flow can be used to create asynchronous digital circuits, and how to create a behavioral test-bench that verifies the correct operation of the asynchronous circuits at all levels of abstraction.

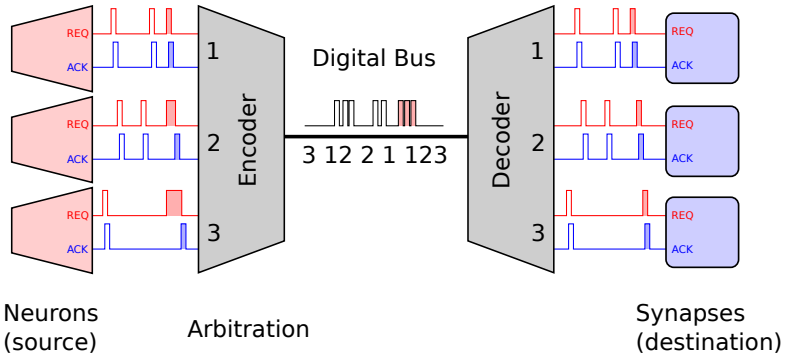


Figure 5.1: AER communication scheme. Neurons send spikes by requesting access (REQ signals) to the arbiter circuit. When the arbiter grants access (ACK signal) the corresponding neurons address is written on a digital bus. A decoder on the receiver side creates a spike for every address on the bus and routes to addressed synapses. The arbiter handles colliding requests (shaded pulses) by sequentially en-queuing them.

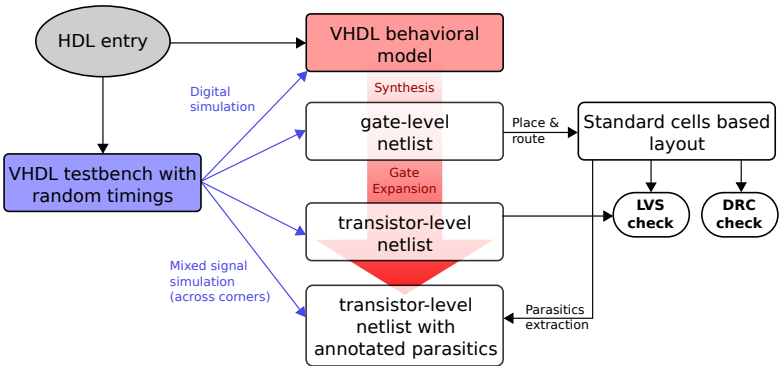
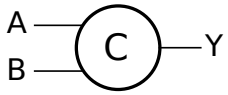


Figure 5.2: Design flow for synchronous circuits that includes transistor level simulations.

5.1.1. Muller-C Element

The Muller-C element (Muller & Bartky, 1959) is a basic building block for many asynchronous circuits. Figure 5.3a shows the truth table and an example of a behavioral VHDL description of the Muller-C element. By providing this VHDL description to the Cadence Encounter RTL compiler synthesis tool, I generated the logic circuit, based on standard cells, shown in Fig 5.3c. The state holding element in this circuit is a latch with an active low asynchronous reset line. In actual implementation I use an additional input, a global reset line, that resets the latch at power-up. As the VHDL code for doing this is straightforward, I omit the description of the power-up reset mechanism. Lets analyze the the circuit of Fig. 5.3c: if initially $Y = 0$, then the latch is enabled (given also that QN is high) and the latch input propagates to the latch output. When the input lines change to $A = 1, B = 1$, the latch output Q changes to 1 and the latch is disabled, by the feedback connection from its complement QN to the latch enable pin E. Note that the '1' signal has to propagate to the latch output before it can disable the latch; so when the latch is disabled, we are guaranteed that $Y = 1$. The only way to change the latch output is through the asynchronous reset line which is activated when $A = 0, B = 0$ and which sets $Y = 0$. It can easily be seen that the output Y remains unchanged when $A = 0, B = 1$ or $A = 1, B = 0$ so the circuit in Fig. 5.3c realizes a Muller-C element using components that are available in standard cell libraries. Due to the unconventional form of the behavioral VHDL

A	B	Y
0	0	0
0	1	no change
1	0	no change
1	1	1

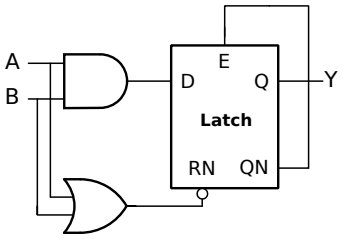


(a)

```
library cadence;
use cadence.attributes.all;

architecture beh of celem is
attribute async_set_reset_process of c1:
label is true;
signal y_s : std_logic;
begin
  y<=y_s;
  c1 : process(a,b,y_s)
   begin
     if a='0' and b='0' then
       y_s<='0';
     elsif y_s='0' then
       y_s<= a and b;
     end if;
   end process;
```

(b)



(c)

Figure 5.3: Muller-C element. (a) Symbol and truth table. (b) Behavioral VHDL code description. (c) Logic circuit automatically generated by the standard logic synthesis tool.

code (Fig. 5.3b), I configured the synthesis tool to use a latch with asynchronous reset by setting the attribute "async_set_reset_process" of the VHDL process describing the Muller-C element to true.

5.1.2. Four-phase Handshake Pipeline Element

The 4-phase handshake cycles of asynchronous circuits are often pipelined to speed-up processing without having to wait for the last part of the data processing chain (e.g., the off-chip receiver) to respond. I describe here the basic building block that can be used to design the complete pipe-lining circuit. This pipeline element is shown in Fig. 5.4. It is inserted between an asynchronous sender and an asynchronous receiver that both implement the 4-phase handshake protocol. The pipeline element can complete up to three phases of the 4-phase handshake cycle with the sender without waiting for any response from the receiver as shown in Fig. 5.4c. By cascading two of these pipeline elements, the full handshake cycle can be completed with the sender without waiting for a receiver response.

5.1.3. Testing Methodology

The asynchronous circuits described in this chapter impose no constraints on the input signals beyond adherence to a specific handshake protocol. The circuits could in principle fail or deadlock for particular timing relations between the input signals. So, in order to verify the robustness of the automatically synthesized circuits I developed a testing methodology that repeatedly samples from the space of all possible input timing relations and checks for the correct operation of the circuit under each sample of the timing relations. This testing methodology is in effect a Monte Carlo method. I illustrate this testing methodology on a circuit composed of two pipeline elements (as the one shown in Fig. 5.4b) connected in cascade. The circuit implements full pipe-lining of the sender requests. The first step is to develop a VHDL driver. Snippets of an example VHDL driver code are shown in Fig. 5.5a. In the code segment on the left, there are instructions that make the driver sample from a uniform distribution between zero and one, and then use this sample to wait a random amount of time, up to one nanosecond, before responding to the circuit under test. The VHDL code on the right contains assertions that signal any

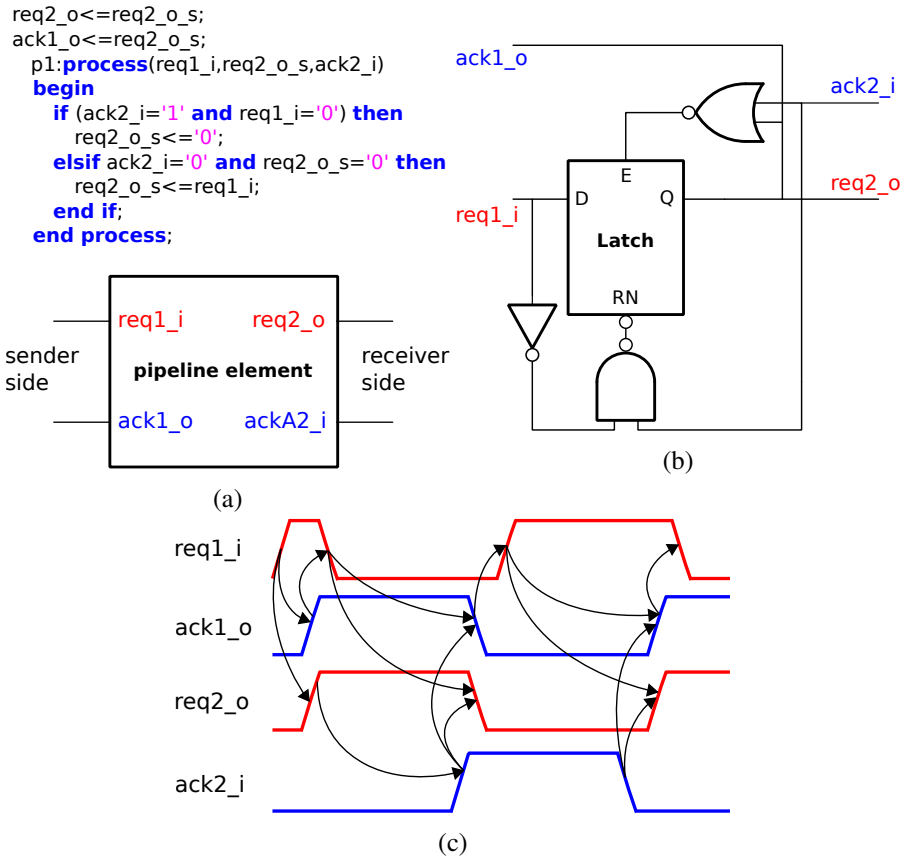


Figure 5.4: Pipeline element. (a) Behavioral VHDL code snippet. The bottom part shows which of the pipeline element signals go to the sender and which signals go to the receiver. (b) Logic circuit generated by standard logic synthesis tools. (c) Signal transition diagram; the pipeline element can complete three phases of the handshake cycle with the sender: $req1\_i\uparrow, ack1\_o\uparrow, req1\_i\downarrow$, before the receiver has responded by pulling "$ack2\_i$" high.

deviation from correct behavior. The same driver code is used to test the design at all levels of abstraction, ranging from behavioral to gate, to transistor level.

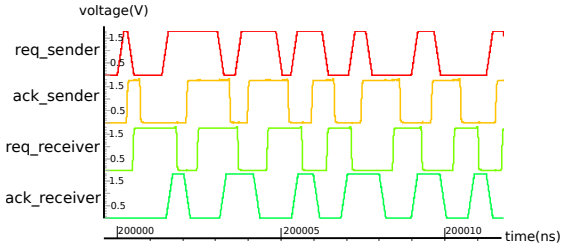
Pure digital simulations are used when testing the behavioral VHDL code and the automatically synthesized gate level netlist. The latter makes use of foundry supplied digital delay models of the gates. Mixed signal simulations using Cadence's AMS designer were used to test the transistor level circuits. Boundary elements (effectively analog-to-digital and digital-to-analog converters) are automatically inserted by the EDA tool to interface the VHDL driver to the transistor level circuit. This testing methodology has many advantages: errors are caught as early as possible in the design process, possibly at the behavioral or gate level description levels; the digital gate level simulations that incorporate gate delays can be run for extremely long times, as they are purely event-based simulations, to uncover even very subtle timing failures; transistor level simulations can also be run for long times without having to save any wave forms for later analysis because the assertions in the VHDL driver monitor the correct operation of the circuit at each point in the simulation.

In Fig. 5.5b, I show wave forms from transistor level simulations of the two cascaded pipeline elements, in which the standard gates of each pipeline element were automatically expanded to transistor level. Notice the random timings of the driver output signals: "req_sender" and

```
req_sender<='1';
 wait until ack_sender='1' for 1.0 us;
 assert ack_sender='1' report "ack error";
uniform(seed1,seed2,R);
 wait for R* 1.0 ns;
req_sender<='0';
```

```
if(rising_edge(req_receiver)) then
 assert ack_receiver='0'  report "req goes high when ack=1";
 end if;
 if(rising_edge(ack_sender)) then
 assert req_sender='1'  report "unsolicited acknowledge";
 end if;
```

(a)



(b)

Figure 5.5: Testing two cascaded pipeline elements. (a) Snippets from the VHDL driver code showing randomized timing of driver output and assertion based verification. (b) Simulation results obtained by driving the transistor level representation using the VHDL driver in a mixed signal simulation

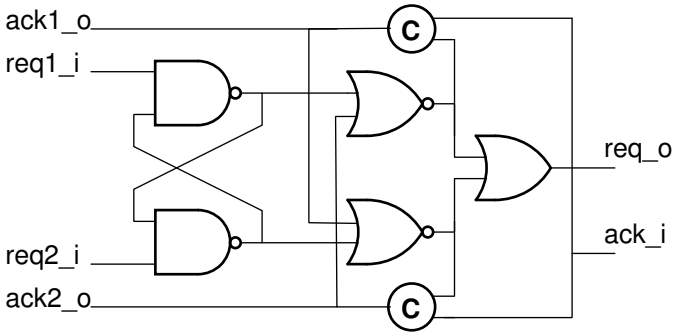


Figure 5.6: Arbiter cell schematic. The logic blocks (C) represent Muller-C elements, which can be implemented using standard gates (see Fig. 5.3).

"ack_receiver", and how up to one complete transaction can be completed with the sender independently of any response from the receiver.

5.2. Output Interface Implementation

An integral part of the output AER interface is the *arbiter* that serializes the access of many asynchronous channels to one shared channel. The arbiter that I implemented is based on the arbiter cell proposed in (Martin & Nystrom, 2006), and shown in Fig. 5.6. The arbiter cell time-domain multiplexes two input non-exclusive asynchronous channels onto a shared output asynchronous channel. This cell can be described using behavioral VHDL code. I make use of the behavioral description of the Muller-C element described in section 5.1.1.

I synthesized a 64 channel output interface, including the arbiter, from behavioral VHDL code using only the logic elements that are found in standard cells libraries. The layout was then created from the gate level netlist using standard digital placement and routing tools. The interface, shown in Fig. 5.7a, is responsible for communicating asynchronous address-events to an off-chip receiver using a 4-phase handshake protocol. It is composed of a 64-channel arbiter tree, shown on the right of Fig. 5.7a. The arbiter tree is built by connecting 63 arbiter cells in

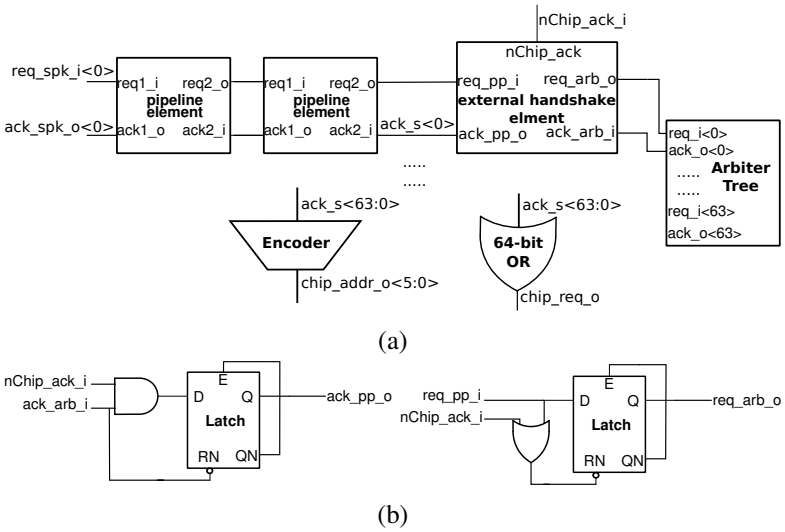


Figure 5.7: Output interface. (a) Block diagram, showing only the first of the 64 input rows. (b) External handshake element.

a binary tree structure. Inputs to the arbiter come from asynchronous and parallel sources of Address-Events (e.g., spiking neurons). Each channel is fully pipelined by passing it through two pipeline elements connected in cascade.

The 64 channels are encoded to form an off-chip asynchronous channel, composed of six data lines "chip_addr_o<5:0>" and of two additional handshaking lines: the request line: "chip_req_o", which is pulled high when an Address Event is generated on the off-chip channel, and the active low acknowledge line coming from the off-chip receiver "nChip_ack_i".

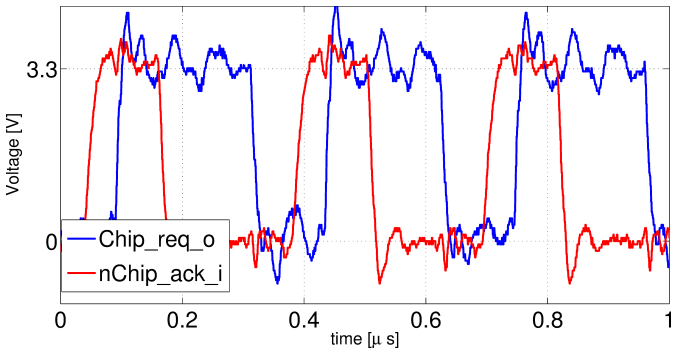
An addition external handshake element, shown in Fig. 5.7b, is required to hold the 4-phase handshake cycle until the address event has been communicated off-chip.

5.3. Experimental Results

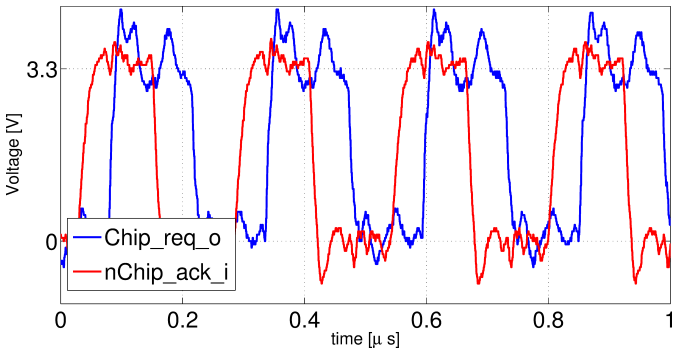
The experimental measurements in this section were obtained by Federico Corradi.

This section presents measurements from a chip fabricated using a $0.18\mu m$ CMOS 1-poly 4-metals technology. The chip is a neuromorphic multi-neuron chip that contains 64 asynchronous spiking elements. The output interface shown in Fig. 5.7a occupies a silicon area of $31377\mu m^2$ (1796 logic gates). I implemented in the chip an additional logic block that allows an external signal to control whether or not to bypass the pipeline stages. The output interface can thus have two modes of operations: pipe-lined and non-pipe-lined modes.

The array of spiking elements was made to fire at high frequencies in order to measure the throughput of the output interface. Off-chip, we delay the "chip_req_o" signal, to ensure that all channel address bits have settled to a valid state on the receiver side, before it goes high. We verified that the produced output data are consistent with the activity in the spiking array by using a logic analyzer. Figure 5.8 shows AER transactions for both pipelined and non-pipelined modes. The pipelined mode has a greater throughput as parts of the handshake cycle between the spiking elements and the off-chip receiver can be completed in parallel.



(a)



(b)

Figure 5.8: Measurements of AER transactions. (a) Non-pipelined mode. (b) Pipelined mode. The throughput, computed from repeated 1 second measurements, is about 25% more in the pipelined versus the non-pipelined mode.

6

Hybrid CMOS-memristor Architecture

This chapter is based on the paper "Implementation of a spike-based Perceptron learning rule using TiO_{2-x} memristors" by Hesham Mostafa, Ali Khiat, Alexander Serb, Christian Mayr, Giacomo Indiveri, and Themis Prodromakis, Frontiers in Neuroscience, 2015

Synapses outnumber neurons by several orders of magnitude in biological neural networks (Binzegger et al., 2004). Reproducing this feature in neuromorphic electronic circuits presents a scaling problem, as integrating thousands of dedicated synapse circuits per neuron can quickly become infeasible for systems that require a large number of neurons (Schemmel et al., 2007). This scaling problem has traditionally been solved by either treating synapses as simple linear elements and time-multiplexing spikes from many pre-synaptic sources onto the same linear circuit (Benjamin et al., 2014), or by treating them as basic binary elements that can be set either "on" or "off" externally, without learning abilities (Merolla et al., 2014b).

Real synapses, however, exhibit non-linear phenomena like spike timing dependent plasticity (STDP) that modulate the weight of an individual synapse based on the activity of the pre- and post-synaptic neurons (Bi & Poo, 1998). The modulation of synaptic weights through plasticity has been shown to greatly increase the range of computations that neural networks can perform (Abbott & Regehr, 2004). Capturing the plasticity properties of real synapses in analog neuromorphic hardware requires the use of distinct physical circuits/elements for each synapse. In conventional CMOS, this can lead to restrictions on scalability. Some potential solutions to the scalability issues in pure CMOS technology involve the use of very large integrated structures (e.g., up to a full wafer (Schemmel et al., 2012)) or the adoption of deep submicron technologies (Noack et al., 2015). Scalability restrictions however can be greatly relaxed if one resorts to compact nano-scale circuit elements that can reproduce the plasticity properties of real synapses.

One potential candidate for these elements is the "memristor". Chua (1971) described the memristor as an element which *behaves somewhat like a nonlinear resistor with memory*. Since HP first linked resistively switching devices with the concept of a memristor (Strukov et al., 2008), work on memristive devices has mostly focused on digital storage and logic functions (Linn et al., 2012; You et al., 2014), but there are also applications as analog/multi-level storage (Moreno et al., 2010; Shuai et al., 2013). In the neuromorphic community, memristors are seen as ideal devices for synapse implementations, as they combine three key functions in one device. Memristors can implement biologically realistic synaptic weight updates, i.e. learning (Jo et al., 2010), they can carry out long term multi-valued weight storage, and they can also communicate weighted pre-synaptic activity to the postsynaptic side (Saighi et al., 2015), significantly relaxing scalability restrictions (Indiveri et al., 2013).

Typically, plasticity in these memristive synapses is evoked by applying specific waveforms to the two terminals of the memristor, with the waveforms aligned to pre- respectively postsynaptic pulses (Jo et al., 2010). The correlation of the waveforms across the memristor in turn implements STDP-like plasticity (Mayr et al., 2012), with the form of the STDP curve defined by the applied wave shape (Serrano-Gotarredona et al., 2013). Both hardware and software models of plasticity based on the basic STDP mechanism are typically chosen primarily for their simplicity (Mayr & Partzsch, 2010). It has been argued however that more elaborate models of plasticity are required to reproduce the experimental evidence obtained from more complex synaptic plasticity experiments in real neural systems, and to implement algorithms that can learn to store and classify correlated patterns (Sjöström et al., 2008; Lisman & Spruston, 2010; Senn & Fusi, 2005).

In this chapter, I present a neuromorphic implementation of one of these extended plasticity models that implements a spike-based Perceptron learning algorithm (Brader et al., 2007), which makes use of both analog CMOS circuits and TiO_{2-x} memristive devices. Compared to the more widely used STDP paradigm, the implementation of this learning algorithm on memristors does not employ the postsynaptic spike timing. Instead, it relies on the correlation of presynaptic spikes with signals derived from the postsynaptic neuron, such as its membrane potential and a measurement of its recent spiking activity. These requirements lead to a novel and quite different approach to the CMOS driver circuits which does not require the generation of temporally long waveforms on the pre- or postsynaptic sides.

In addition to spike timing, plasticity in biological synapses also depends on the firing rate of the post-synaptic neuron (Sjöström et al., 2001), a phenomenon that can not be captured by pair-wise STDP mechanisms (Pfister et al., 2006). The spike-based perceptron learning rule explicitly contains a term that reflects the recent firing rate of the neuron and is thus able to realize the rate-dependence of synaptic weight updates. The rule is also able to realize weight updates that depend on pre-post spike timing even though it does not explicitly depend on the post-synaptic spike times. Instead, it uses the membrane potential of the post-synaptic neuron as an indirect estimator of post-synaptic firing times. The rule is thus able to reasonably match the behavior of biological synapses while having a functional form that can be implemented efficiently on pure CMOS or on hybrid CMOS-memristor neuromorphic systems.

6.1. The Plasticity Model

The spike-based Perceptron learning model of long-term plasticity has been introduced in Brader et al. (2007) based on earlier work in Fusi et al. (2000). The model represents a synapse with two stable weights, potentiated and depressed, whereby the transition between the two stable weights is done in an analog or graded manner. The synaptic weight $X(t)$ is influenced by a combination of pre- and post-synaptic activity, namely the pre-synaptic spike time t_{pre} and the value of the post-synaptic neuron membrane voltage $V_{mem}(t)$ and intra-cellular calcium concentration $C(t)$. A pre-synaptic spike arriving at t_{pre} reads the instantaneous post-synaptic values $V_{mem}(t_{pre})$ and $C(t_{pre})$. The change in $X(t)$ depends on these instantaneous values in the following way:

$$\mathrm{X} \rightarrow \mathrm{X} + a \qquad if \qquad \{V_{mem}(t_{pre}) > \theta_V \quad and \quad \theta^l_{up} < C(t_{pre}) < \theta^h_{up}\} \tag{6.1}$$

$$\mathrm{X} \rightarrow \mathrm{X} - b \qquad if \qquad \{V_{mem}(t_{pre}) \leq \theta_V \quad and \quad \theta^l_{down} < C(t_{pre}) < \theta^h_{down}\}, \tag{6.2}$$

where a and b are jump sizes and θ_V is a voltage threshold. In other words, $X(t)$ is increased if $V_{mem}(t)$ is elevated (above θ_V) when the pre-synaptic spike arrives and decreased when $V_{mem}(t)$ is low at time t_{pre} provided that the calcium variable $C(t)$ is in the correct range. θ^l_{up}, θ^h_{up}, θ^l_{down}, and θ^h_{down} are thresholds on the calcium variable. The calcium variable $C(t)$ is an auxiliary variable that is a low-pass filtered version of the post-synaptic spikes (see Brader et al. (2007) for details). The variable $C(t)$ is incremented by J_C at each post-synaptic spike time t_i, where J_C reflects the magnitude of spike-triggered calcium influx into the cell. $C(t)$ decays with a time constant τ_C:

$$\frac{d\mathrm{C(t)}}{dt} = -\frac{1}{\tau_\mathrm{C}}\mathrm{C(t)} + J_\mathrm{C}\sum_i \delta(t - t_i) \tag{6.3}$$

The dependence of the weight updates on $C(t)$ allows the learning rule to enable/disable the weight updates based on the long-term average of post-synaptic activity. $X(t)$ continuously drifts towards one of two stable values based on whether it is above or below the threshold θ_X:

$$\frac{d\mathrm{X}}{dt} = \alpha \qquad if \qquad \mathrm{X} > \theta_X \tag{6.4}$$

$$\frac{d\mathrm{X}}{dt} = -\beta \qquad if \qquad \mathrm{X} \leq \theta_X \tag{6.5}$$

The weight $X(t)$ is bounded above and below by the two stable states X_{high} and X_{low} which are not shown in the equations to simplify the notation. Figure 6.1 illustrates the relevant waveforms and parameters of the spike-based Perceptron learning rule.

The dynamics of the membrane potential variable, V_{mem}, which is used in Eqs. 6.1 and 6.2, depend on the neuron model used. The original neuron model used with the perceptron-learning rule is the simple constant leak integrate and fire neuron model (Brader et al., 2007). The neuron circuit I have in our neuromorphic chip, however, implements the more realistic adaptive exponential integrate and fire neuron model. This neuron circuit and the underlying model are described in detail in Indiveri et al. (2011) and Qiao et al. (2015). The interaction between this adaptive exponential integrate and fire silicon neuron and the spike-based perceptron-learning rule is described in Indiveri et al. (2010)

Although the spike-based plasticity rule described above has been shown to reproduce, on average, the classical STDP phenomenology (Brader et al., 2007), it differs from the vast majority of spike-timing plasticity rules in that it does not explicitly depend on the precise timing of both pre- and post-synaptic neuron spikes. The compatibility with the classical STDP learning rule comes about through the rule's dependence on the post-synaptic neuron's membrane potential: a pre-synaptic spike that occurs when the post-synaptic membrane potential is high will potentiate the synapse and will likely produce a post-synaptic spike shortly after. Thus, the synapse tends to get potentiated in pre-before-post scenarios. The synapse also tends to get depressed

in post-before-pre scenarios because the membrane potential is usually low for a few milliseconds after a post-synaptic spike is emitted, and a pre-synaptic spike arriving in this interval will depress the synapse.

The spike-based Perceptron plasticity rule also has access to post-synaptic neuron's rate information through the $C(t)$ signal. This allows it to reproduce effects beyond classical pair-wise STDP such as increased potentiation at high post-synaptic firing rates and increased depression at low post-synaptic firing rates (Sjöström et al., 2001). These effects arise in more complicated STDP models such as triplet STDP (Pfister et al., 2006; Mayr & Partzsch, 2010). The absence of explicit dependence on the post-synaptic neuron's firing times thus does not diminish the biological plausibility or the computational power of the spike-based Perceptron learning rule.

For the purpose of pure CMOS VLSI implementation(Chicca et al., 2014), this plasticity model is interesting because it can learn a graded response to an input pattern but on long time scales, the weight $X(t)$ drifts to one of two stable states and is thus easy to store long-term. In the hybrid CMOS-memristor architecture that I propose in this chapter, however, the weight drift (Eqs. 6.4 and 6.5) is not implemented. The memristor conductance (weight) only changes on pre-synaptic spikes. Weight drift or the bi-stable synaptic dynamics of the perceptron learning rule can be useful in consolidating the synaptic changes and making the synaptic weight more robust against spurious spikes (Brader et al., 2007). However, this comes at the cost of the sensitivity of the plasticity rule to the temporal spike patterns as multiple spike patterns might lead to the same binary synaptic weights. In the absence of weight drift as in the proposed hybrid CMOS-memristor architecture, the analog synaptic weights are able to maintain a synaptic trace that better reflects the identity of past spiking patterns (Maass & Markram, 2002).

6.2. Circuits for Memristive Learning

The basic building block of the CMOS circuits is a neuron tile which is shown schematically in Fig. 6.2a. The tile contains an analog subthreshold leaky integrate and fire neuron which is fully described in (Qiao et al., 2015). The neuron integrates synaptic current (with an adjustable leak) on a capacitor. When the capacitor voltage crosses the firing threshold, the neuron generates a digital spike and the capacitor voltage is reset to ground. The plasticity circuit monitors the membrane potential, $Vmem$, and the spike output of the neuron and uses them to evaluate the conditions in Eqs. 6.1 and 6.2. The plasticity circuit internally generates the $C(t)$ signal by low-pass filtering the neuron spikes. The plasticity circuit then generates two digital signals: 'up' and 'dn' that determine whether incoming synapses/memristors should be potentiated, depressed, or left unchanged when a pre-synaptic spike occurs according to Eqs. 6.1 and 6.2. The plasticity circuit is described in more detail in (Qiao et al., 2015).

A neuron tile has a pre-synaptic and a post-synaptic memristor terminal. These terminals are monitored and driven by the high voltage post- and pre- interfaces which run at a supply voltage of $5V$. All other circuits operate using a $1.8V$ supply. The $5V$ operation allows the memristor interface circuits to apply higher voltage pulses to the memristor terminals. The memristor conductance changes if pulses above a certain magnitude (the write threshold) are applied across it. The direction of the change depends on the polarity of the pulse. I designed the interface circuits so that they can interface to memristors having resistance values as low as $1\,KOhm$ and deliver write pulses of either polarity with an amplitude of up to $2V$. The write voltage threshold for the memristor devices is much lower than $2V$. The height of the write pulses are programmable, however, so I can control their amplitudes up to $2V$. The width of the programming pulse is also configurable and can be as wide as $1\,ms$. The read pulse amplitude (which needs to be below the write threshold) is adjustable in the $0 - 2V$ range and its width

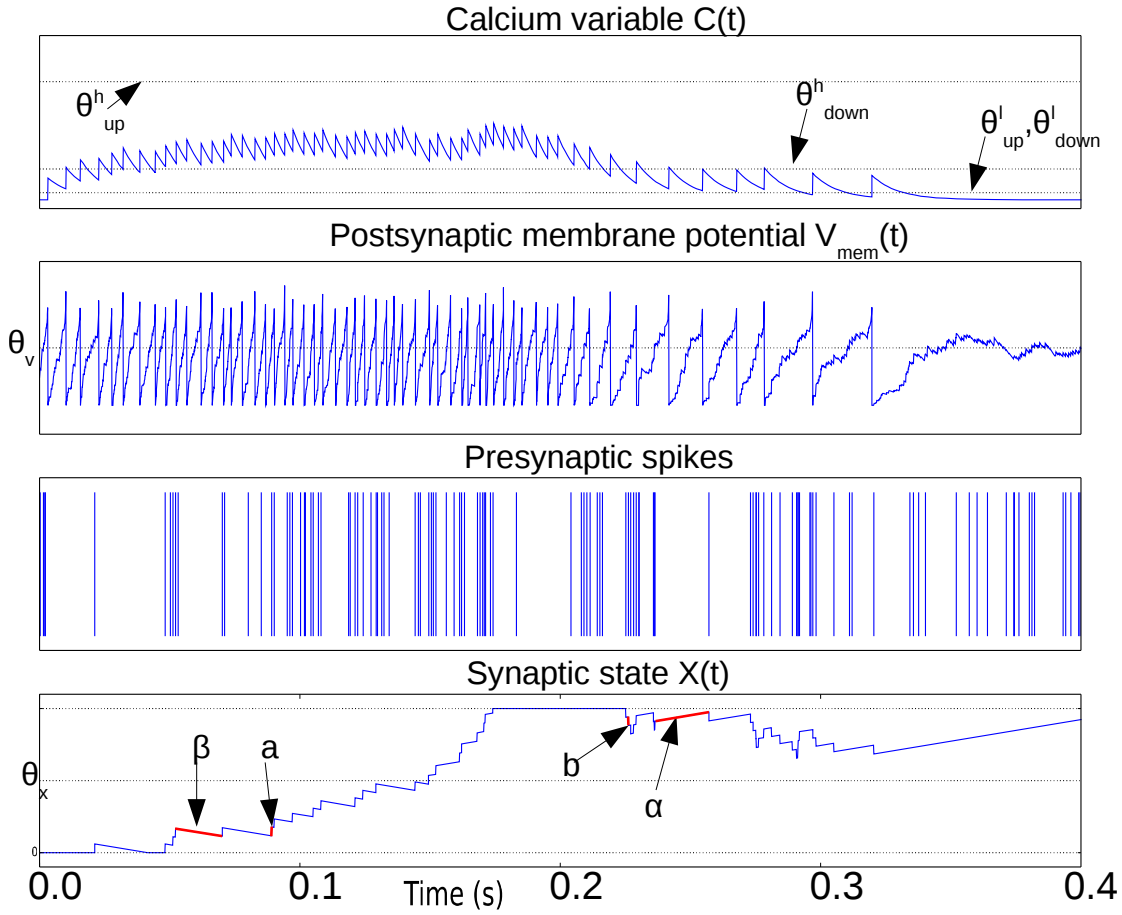


Figure 6.1: Illustration waveforms of the spike-based perceptron learning rule showing key parameters from Eqs. 6.1- 6.5. The Calcium variable plot shows the ranges defined by $\theta^l_{up}, \theta^h_{up}, \theta^l_{down}, \theta^h_{down}$ within which synaptic plasticity is active according to Eqs. 6.1 and 6.2. The post-synaptic neuron membrane potential plot shows the threshold θ_V. Incoming synapses can be depressed (potentiated) if $V_{mem}(t)$ is below (above) θ_V. The bottom plot showing the synaptic state $X(t)$ illustrates the jump and drift mechanism. On each pre-synaptic spike, the mutually exclusive conditions in Eqs. 6.1 and 6.2 are evaluated. If the condition in Eq. 6.1 (Eq. 6.2) is fulfilled, the synaptic state jumps up (down) by a step a (b). The synaptic state is continuously drifting to a high or low state depending on whether it is above or below the threshold θ_X, respectively.

is also adjustable . The memristor is inserted between the pre-synaptic terminal of one tile and the post-synaptic terminal of another (or the same) tile. Spikes generated in the neuron circuit of the pre-synaptic tile will then cause a current proportional to the memristor conductance to be injected into the post-synaptic tile neuron. Moreover, based on the output of the plasticity circuit in the post-synaptic tile, a voltage pulse of the appropriate polarity is applied across the memristor terminals to increase/potentiate or decrease/depress its conductance when the pre-synaptic neuron tile generates a spike. In the rest of this section, I describe how this behavior is realized.

The pre- and post-synaptic memristor interfaces are shown in more detail in Fig. 6.2b where they are linked by a memristive element. I retain the ability to disconnect the post-synaptic

plasticity signal	Vpost	plasticity event	open switches	closed switches
'up'=0 and 'dn'=0	3.0	no change	S4,S5,S7	S6,S8
'up'=1 and 'dn'=0	4.5	potentiate	S4,S6,S7	S5,S8
'up'=0 and 'dn'=1	0.5	depress	S8,S5,S6	S4,S7

Table 6.1: Effect of 'up' and 'dn' signals on the post-synaptic terminal potential which in turn determines the type of plasticity event induced on pre-synaptic spikes. Shown are the open and closed switches in each case. The switches are controlled by the 'up' and 'dn' signals.

circuit from the memristor post-synaptic terminal using switch S1. The pre-synaptic memristor terminal is kept floating by default so no current can flow through the memristor and its value remains constant. The post-synaptic terminal is monitoring the current flowing through the memristor and injecting a proportional current into the neuron. By keeping the pre-synaptic terminal floating, no current flows through the memristor, and no current is injected into the post-synaptic neuron. When the pre-synaptic tile neuron spikes, or when the tile receives an AER event from off-chip, the pre-synaptic terminal is strongly clamped at $2.5V$ for a short duration that is controlled by an analog bias. By clamping the pre-synaptic terminal to the middle of the supply voltage, I am able to apply pulses of either polarity with an amplitude of up to $2.5V$ by setting the appropriate voltage on the post-synaptic terminal. If the post-synaptic terminal is clamped to V_{post}, then on pre-synaptic spike, a pulse of amplitude $V_{post} - 2.5$ is applied across the memristor. Assume switch S1 is closed. The post-synaptic terminal can be clamped to one of three possible values: 4.5V, 0.5V, or 3V. These clamping voltages can be adjusted through analog biases. The clamping is done by the strong transistors M1 and M2 which are each part of a negative feedback loop that controls their gate potentials so as to maintain their drain potentials at one of the three voltage clamp values. A number of switches which are controlled by the 'up' and 'dn' signals from the plasticity circuit determine which clamping voltage is selected according to Table 6.1. For example, if switches S5 and S8 are closed and switches S4, S6, and S7 are open, the post-synaptic terminal is clamped by a PFET at 4.5V. Switches S4-S8 are implemented as single transistors as each switch has to pass a bias voltages that is always either above 2.5 (PFET is used) or below 2.5 (NFET is used). Switch S1 is implemented as a transmission gate.

At a pre-synaptic spike which causes the pre-synaptic terminal to be clamped to $2.5V$, the memristor experiences a voltage pulse of either 2.0V, -2.0V, or 0.5V depending on whether the post-synaptic terminal is at 4.5V, 0.5V, or 3V respectively. These three cases can either potentiate/increase the memristor conductance, depress/decrease it, or leave it unchanged respectively. It is the plasticity circuit, which through the 'up' and 'dn' signals controls switches S4-S8, which chooses between these three cases (table 6.1).

The post-synaptic side indirectly senses the memristor conductance from the gate voltages V1 and V2. When the pre-synaptic side is floating, the two feedback loops push V1 and V2 to 5V and 0V respectively. The current generation circuit will then generate very little current. At a pre-synaptic event, either V1 or V2 abruptly changes so that the actively clamping transistor has increased effective gate-source voltage so as to be able to source/sink the memristor current while maintaining the drain terminal at the clamp voltage. The change in V1 or V2 is proportional to the memristor conductance and based on this change, a proportional current $Isyn$ is generated and injected into the post-synaptic neuron. The current generation circuit

approximately implements the equation:

$$Isyn = A * V2 - B * V1$$

Where A and B are constants adjusted through biases. This linear equation is, however, valid in a limited regime of V_1 and V_2. This regime can be adjusted through biases. Note that $Isyn$ is proportional to the absolute value of the memristor current, regardless of whether the current is sourced by transistor M1 or sunk by transistor M2. The 'up' and 'dn' signals can not both be high at the same time. For the three possible configurations of the 'up' and 'dn' signals in table 6.1, M1 and M2 can not be supplying current at the same time. For the possible configuration of switches shown in table 6.1, the feedback loops controlling V1 and V2 ensure that the gate-source voltage of M1 and M2 can not be simultaneously non-zero. This guarantees that the current in either M1 or M2 is the current flowing through the memristor.

This active clamp technique allows maximum voltage headroom for transistors M1 and M2 which allows them to clamp the post-synaptic terminal at voltages near the supply rails. It also enables precise control over the magnitude of the voltage pulses applied across the memristor. In section 6.3, I present experimental results to illustrate the behavior of the circuit in Fig. 6.2a.

The synaptic weight in the original spike-based Perceptron learning rule has only two stable states due to the weight drift (Eqs 6.4 and 6.5) which pushes the weight to either a high or a low value. This mechanism is not present in our architecture; the synaptic weight (memristor conductance) is an analog quantity that only changes in response to pre-synaptic spikes and is stable otherwise. Realizing analog synaptic weights that are long-term stable is difficult in pure CMOS as analog weights that are encoded using charge on a capacitor are easily corruptible through leakage paths and capacitive coupling to nearby nodes. Therefore, in pure CMOS, a multi-stability mechanism is required to push the weights to well-defined and stable discrete states. Hybrid CMOS-memristor architectures like ours can realize naturally stable analog weights (memristor conductances) and thus do not require such a mechanism.

An 8*8 array of the neuron tile shown in Fig. 6.2 was fabricated on a standard 6M 180 nm CMOS process as part of a larger multi-purpose neuromorphic chip shown in Fig. 6.3a. The chip contains a bias generator based on the design in Delbruck & Lichtsteiner (2006). The bias generator has a low pin-count (5 pins) digital programming interface that can be used to set the values of the analog biases used in the neuron tile array. The other components of the multi-purpose neuromorphic chip are described in detail in (Qiao et al., 2015) but they are not relevant for the current chapter. Address event representation (AER) interfaces carry spikes to/from the neuron tile array. The pre- and post-synaptic terminals of the 64 neuron tiles were routed to the top-metal level to make it possible to directly deposit a cross-bar array of TiO_{2-x} memristors on top that connects a memristor between each pre-synaptic terminal and each post-synaptic terminal as shown in Fig. 6.3b. This post-processing step was not carried out. In the chip, the pre- and post-synaptic memristor terminals of two neuron tiles were directly connected to pads. An off-chip memristor was then connected between the pre-synaptic terminal of one of these neuron tiles and the post-synaptic terminal of the other tile as shown in Fig. 6.3b. This setup was used to obtain the measurements presented in the rest of this chapter.

6.3. Characterization of CMOS Plasticity Circuits

On each pre-synaptic spike, the pre- and post-synaptic interface circuits in the neuron tile shown in Fig. 6.2 apply a voltage pulse to update the memristor value according to the spike-based perceptron learning rule. This behavior is illustrated in Fig. 6.4a where a fixed resistor was inserted between the pre-synaptic terminal of one tile and the post-synaptic terminal of another (as in

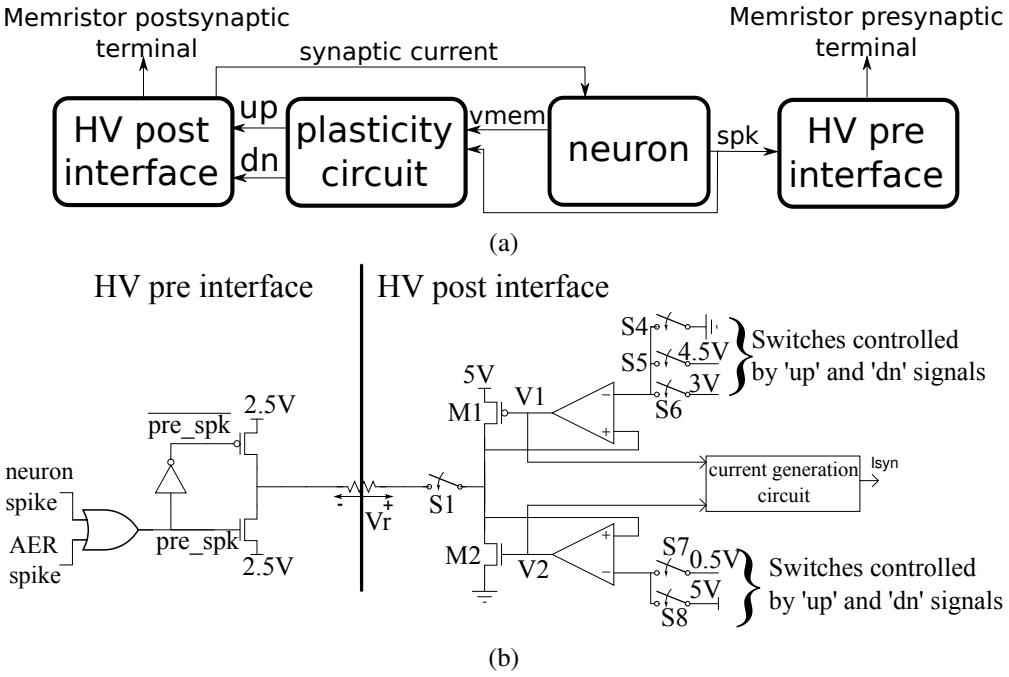


Figure 6.2: (a) High-level schematic of a neuron tile. (b) More detailed schematics of the high voltage pre- and post-synaptic interfaces in the neuron tile.

Fig. 6.3b but using a resistor instead of a memristor). Constant current is injected into the neurons to maintain a constant firing rate. The calcium signal, $C(t)$, jumps up after each spike and enters the plasticity range, then it decays back out of the plasticity range. The bottom two plots in Fig. 6.4a show the 'up' and 'dn' signals and the neuron membrane potential in the post-synaptic tile. The 'up' and 'dn' signals are generated by the plasticity circuit in the post-synaptic tile (see Fig. 6.2a). This plasticity circuit calculates the calcium variable, $C(t)$, from the post-synaptic neuron spikes according to Eq. 6.3. It evaluates the conditions in Eqs. 6.1 and 6.2 to decide whether to potentiate, depress, or leave unchanged incoming synapses when the pre-synaptic neuron spikes. This decision is communicated to the post-synaptic interface circuit which clamps the post-synaptic terminal voltage $vpost$ at $4.1V$ (when the 'up' signal is high), $0.1V$ (when the 'dn' signal is high), or $3V$ (when both the 'dn' and 'up' are low) as shown in Fig. 6.4a . The pre-synaptic terminal is floating by default and is clamped at $2.5V$ for a short duration on each pre-synaptic spike. For each pre-synaptic spike, this causes $vpost - vpre$ to be approximately $2V$ when the 'up' signal is high which would increase the memristor conductance (potentiation), $-2V$ when the 'dn' signal is high which would decrease the memristor conductance (depression), and $0.5V$ otherwise as shown in Fig. 6.4a which would leave the memristor conductance unchanged and simply read out its value. In Fig. 6.4a, at the first pre-synaptic spike, $C(t)$ is outside the plasticity range and a small read pulse is applied. The subsequent pre-synaptic spikes arrive first in the depression, then in the potentiation intervals of the post-synaptic tile and large amplitude pulses with the appropriate polarity are applied.

Fig. 6.4b shows how the firing frequency of the post-synaptic neuron varies as a function of the value of the resistor connecting it to the pre-synaptic tile. The pre-synaptic tile generates spikes at a constant frequency. The firing frequency of the post-synaptic neuron steadily decreases with the decreasing conductance of the resistor. The bias conditions were chosen to obtain a linear region in the 1 KOhm to 4 KOhm resistance range beyond which the post-synaptic neuron firing frequency saturates at a lower bound. The neuron is biased to have a

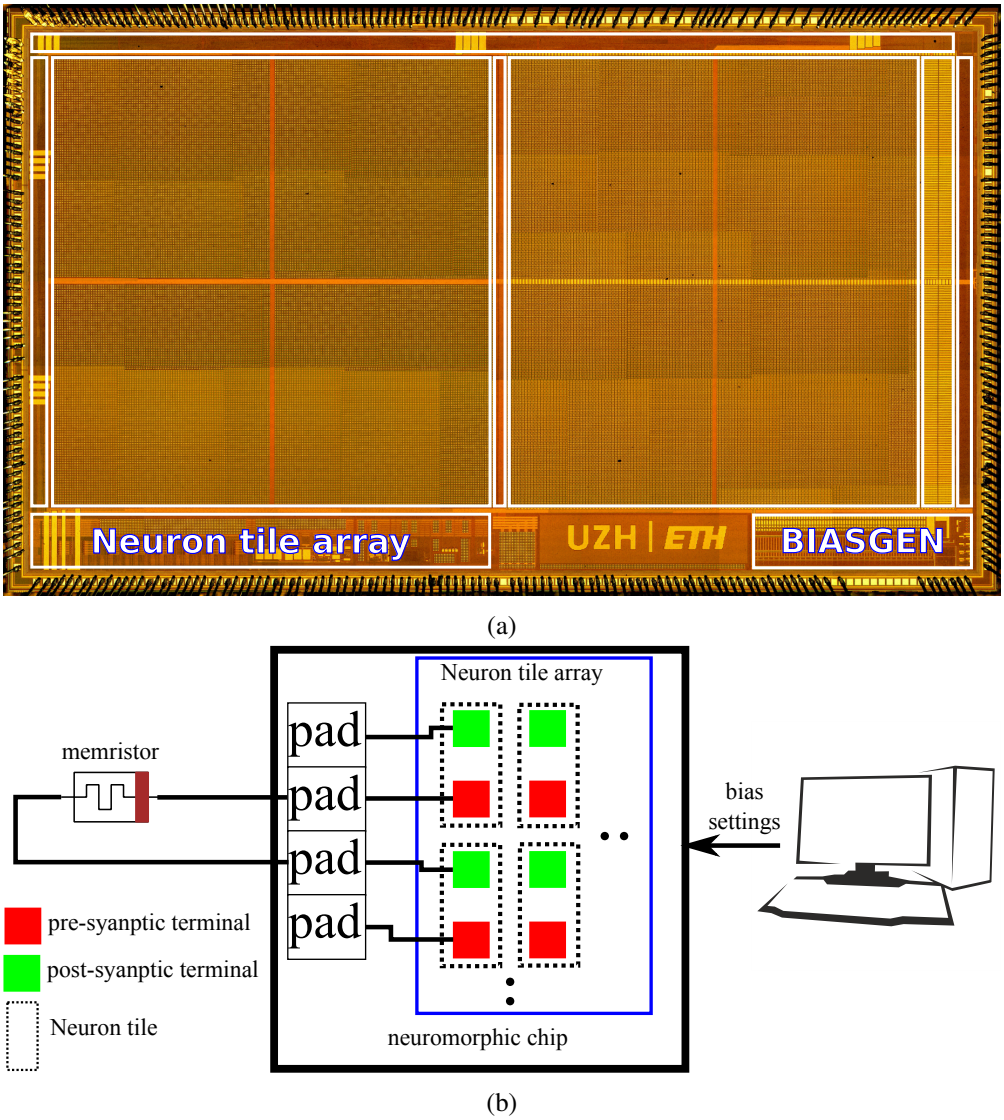
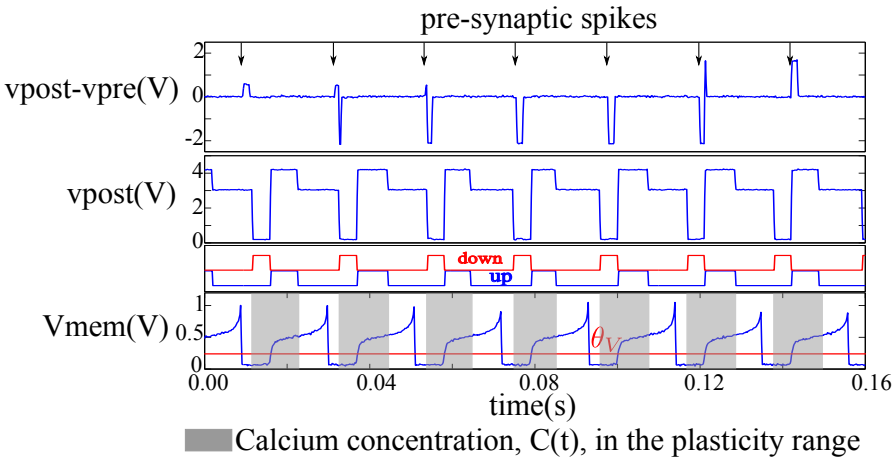
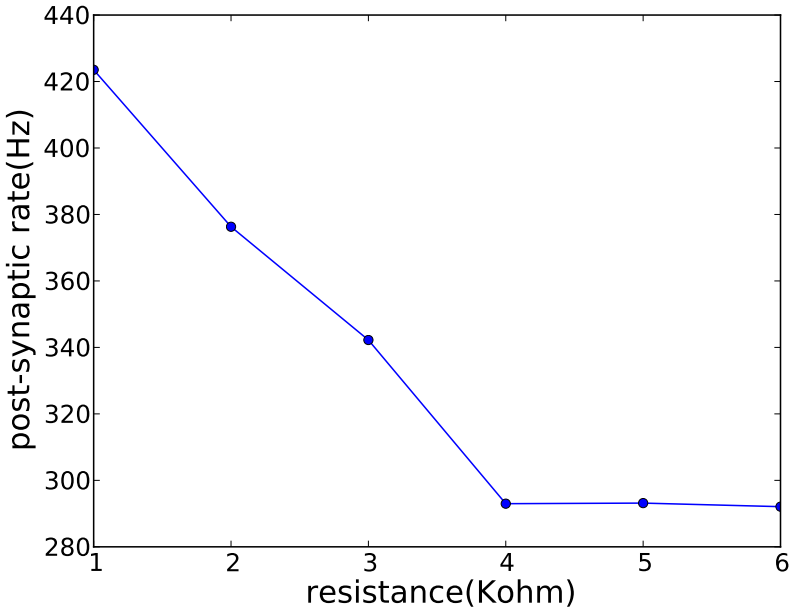


Figure 6.3: (a) Micrograph of the multi-purpose neuromorphic chip die showing the neuron tile array and the bias generator. (b) Illustration of the hardware setup used to obtain measurements for this chapter. The pre-synaptic terminal of one neuron tile is connected to the post-synaptic terminal of another neuron tile through an off-chip TiO_{2-x} memristor. A PC controls the digital settings of the on-chip bias generator.

spontaneous baseline firing rate which is about 290Hz. The transfer function from the synaptic resistance to the synaptic current injected into the neuron is linear in the 1KOhm to 4KOhm region in Fig. 6.4b but beyond that, it is highly non-linear causing a small increase in synaptic resistance to lead to a greatly reduced synaptic current which becomes negligible compared to the constant injection current used to maintain the baseline firing rate. The neuron then saturates at the baseline firing rate.



(a)



(b)

Figure 6.4: (a) Membrane potential and the 'up' and 'dn' signals of a post-synaptic tile receiving spikes at a constant rate from a pre-synaptic tile. The plasticity circuit controls the post-synaptic terminal potential so as to generate an appropriate programming/read pulse across the $vpost-vpre$ terminals on each pre-synaptic spikes. (b) Decrease of post-synaptic neuron firing frequency as the conductance of the afferent synaptic element decreases while the pre-synaptic neuron firing rate is kept constant.

6.4. Memristive Plasticity Experiments

In the fabricated chip, the pre- and post-synaptic terminals of two neuron tiles were available on the chip pads. The pre-synaptic terminal of one tile was connected to the post-synaptic terminal of the other tile through a TiO_{2-x} memristor as shown in Fig. 6.3b. The pre-synaptic neuron was biased to fire at $47\,Hz$. The spike-based perceptron learning circuit was then suc-

cessively cycled between the potentiation and depression regimes (using the analog biases) and the resulting post-synaptic firing rate was observed. The post-synaptic firing rate was taken as an indication of the synaptic weight or the conductance of the memristive element. The system showed correct operation with the postsynaptic firing rate increasing after a potentiation episode and decreasing after a depression episode as shown in Fig. 6.5.

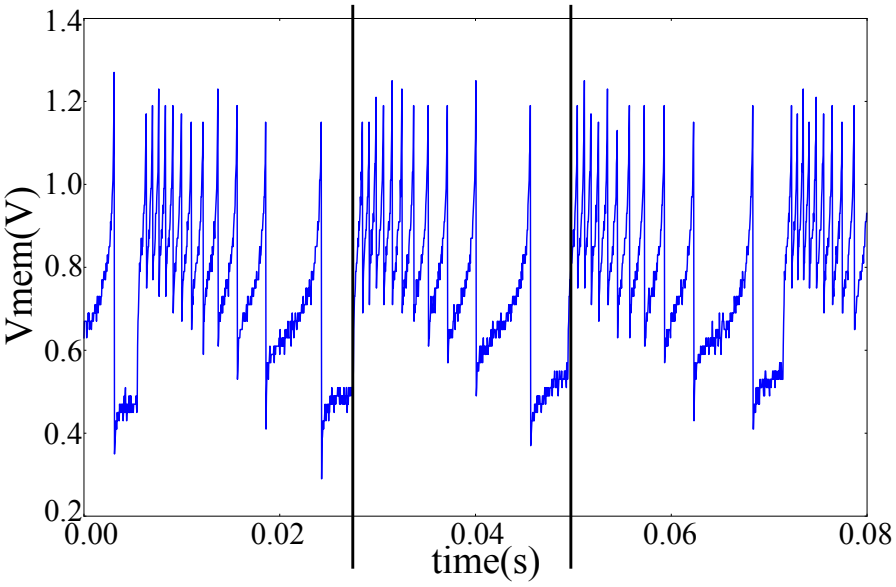
A potentiation episode involves setting the biases of the plasticity circuit in Fig. 6.2a so that its 'up' output signal is constantly high which will cause the memristor post-synaptic terminal to be clamped at approximately $4.1V$ as shown in Fig. 6.4a. On each pre-synaptic spike, a pulse of approximately $2V$ is thus applied across the memristor terminals which will act to increase its conductance. Similarly, in a depression episode, the 'dn' output signal is constantly high which will cause a pulse of approximately $-2V$ to be applied across the memristor terminals on each pre-synaptic spike which will act to decrease its conductance. After each plasticity episode, plasticity was disabled and the post-synaptic neuron firing rate measured, then the next plasticity episode is applied.

6.5. Discussion

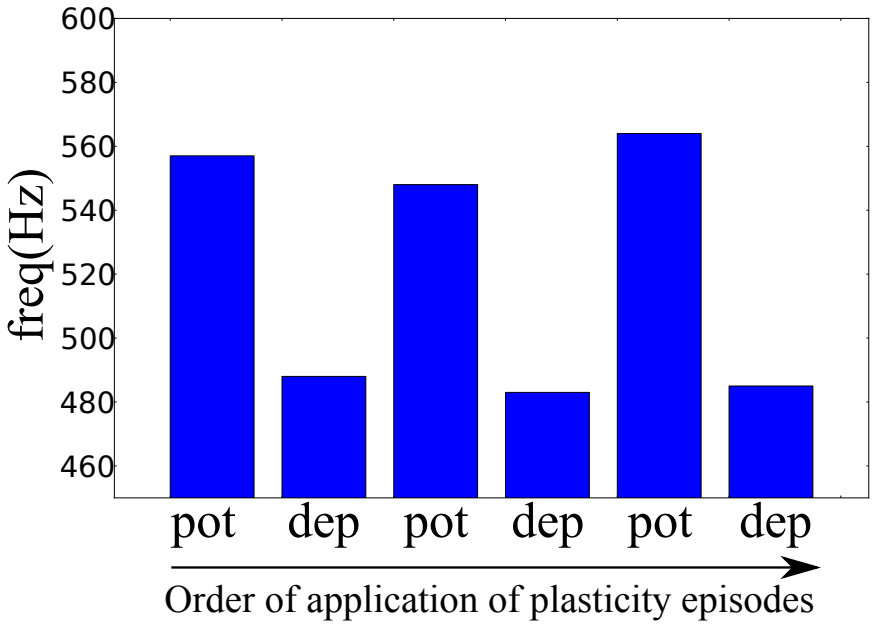
The spike-based Perceptron plasticity rule has been implemented in CMOS neuromorphic systems using various types of circuits such as subthreshold circuits (Mitra et al., 2009) and switched capacitor circuits (Noack et al., 2015). In this chapter, I presented a physical implementation of the first hybrid CMOS-memristor architecture that implements a spike-based Perceptron learning plasticity rule. The physical CMOS-memristor system I presented is a standalone system in which the custom CMOS chip connects directly to the memristive devices. The CMOS chip implements the neuron elements together with dedicated per-neuron circuits that can program (potentiate or depress) the memristive synaptic elements as well as sense their conductances/weights to generate proportional Excitatory Post-Synaptic Currents (EPSCs) in the post-synaptic neuron in response to pre-synaptic spikes. I have presented direct measurements that illustrate the behavior of this physical CMOS-memristor system. This is the first standalone neuromorphic system that combines custom neuron circuits with memristor programming and sensing circuits acting on physical memristive devices.

Many highly accurate and biologically grounded, i.e, non-empirical, synaptic plasticity rules make use of several auxiliary variables beyond spike times in the pre- and post-synaptic neurons to control synaptic weight updates (Clopath & Gerstner, 2010; Graupner & Brunel, 2012; Pfister et al., 2006; Brader et al., 2007; Mayr & Partzsch, 2010). These auxiliary variables may include low-pass filtered versions of the membrane potential (Clopath & Gerstner, 2010) or a low-pass filtered version of the neuron's spike train. (Brader et al., 2007). Interestingly, the time difference between pre- and post-synaptic spikes does not figure explicitly in these models. This presents a problem for current neuromorphic memristive architectures that mainly depend on this time difference (through the overlap between between pre- and post-synaptic spike-triggered waveforms) to induce weight updates. These architectures will not be able to handle weight updates that are triggered on single pre- or post- synaptic spikes.

The architecture I presented triggers weight updates on single pre-synaptic spikes. This has a significant advantage: at the time of a pre-synaptic spike, the neuromorphic synapse can be immediately potentiated or depressed based on the current state of the post-synaptic neuron; the neuromorphic system does not have to wait for a post-synaptic spike to know the outcome of the plasticity event. Implementations of classical pair-wise STDP rules using memristors typically trigger long waveforms on the pre- and post-synaptic sides of the memristor in response to pre- and post-synaptic spikes respectively. When these waveforms overlap, the potential difference



(a)



(b)

Figure 6.5: (a) The post-synaptic membrane potential in response to a pre-synaptic pulse train of $47\,Hz$. Vertical lines indicate the times of two pre-synaptic spikes. (b) Frequency of the post-synaptic neuron after a number of plasticity episodes. Plasticity episodes were applied in succession by adjusting the analog biases to put the post-synaptic neuron tile in the potentiation and depression regimes.

across the memristor exceeds a threshold and changes in memristor conductance occur. The duration of these waveforms dictate the STDP window. The overlapping waveforms paradigm is problematic in the high spike rate regime as multiple spikes can occur within the STDP window, thereby corrupting the synaptic weight update. By contrast this problem is completely avoided in the case of the spike-based Perceptron learning rule.

In the original learning rule (Brader et al., 2007) the weights were bistable, i.e, they gradually drifted to one of two stable states. This had the effect of consolidating synaptic changes and making it more difficult for a synaptic pattern to be corrupted by spurious spikes. Our architecture does not implement such continuous (non event-driven) weight drift. This indicates that synaptic rule features that simplify pure CMOS implementations like bistable weights do not necessarily translate to simpler CMOS-memristor implementations.

In the next chapter, I return to the main topic of this thesis. I describe a novel hardware architecture inspired by the biologically-motivated oscillatory networks presented in chapters 3 and 4 which can be be used to efficiently solve CSPs.

7

The InferenceEngine Architecture

This chapter is based on the paper "An event-based architecture for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Muller, and Giacomo Indiveri, Nature Communications, Accepted.

Constraint satisfaction problems (CSPs) are a fundamental class of problems in computer science with wide applicability in areas such as channel coding (MacKay, 2003), circuit optimization (Kirkpatrick et al., 1983), and scheduling (Garey et al., 1976). Algorithms for solving CSPs are typically run on classical von Neumann computing platforms that were not explicitly designed for these types of problems. This chapter addresses the question: how can we implement a more efficient computing substrate whose architecture and dynamics better reflect the distributed nature of CSPs?

Many dynamical systems that have been proposed for solving CSPs violate the 'physical implementability' condition (Zhang & Constantinides, 1992; Nagamatu & Yanaru, 1996; Ercsey-Ravasz & Toroczkai, 2011). Non-physicality arises from the use of variables that can grow without bounds as the system is searching for solutions. On the other hand, there is a long, well-established tradition of studying physically realizable dynamical systems, e.g., in the form of artificial neural networks, to solve CSPs or "best-match problems" (Minsky & Papert, 1969; Rumelhart & McClelland, 1986). Early attempts in this field used attractor networks, such as Hopfield networks (Hopfield, 1982), to solve NP-hard problems like the traveling salesman problem (Hopfield & Tank, 1985, 1986). These attractor networks, however, would often get stuck at locally optimal solutions. To overcome this problem, stochastic mechanisms were proposed (Habenschuss et al., 2013; Maass, 2014) which require explicit sources of noise to force the network to continuously explore the solution space. While noise is an inextricable part of any physical system, dynamically controlling its power to balance "exploratory" versus "greedy" search, or to move the network from an exploratory phase to a greedy one according to an annealing schedule, is not a trivial operation and puts an additional overhead on the physical implementation.

In this chapter, I present a mixed analog/digital hardware architecture whose dynamics execute an efficient search for CSP solutions without the need for external sources of noise. For this reason the architecture can be easily and efficiently implemented using Complementary Metal-Oxide Semiconductor (CMOS) Very Large Scale Integration (VLSI) electronic circuits. In the proposed architecture, each variable in a CSP is represented by a node consisting of an analog oscillator and a state-holding asynchronous digital circuit. To achieve robust and scalable computation, the nodes communicate using digital pulses, or events. This combination of analog and digital circuits running in a hybrid continuous/event-driven mode avoids many of the problems that affect pure analog VLSI systems such as susceptibility to noise, degradation of analog

signals during storage and communication, and signal restoration/refresh issues. Since the oscillators are realized using analog circuit, when fabricated, they exhibit incommensurable natural frequencies, i.e., frequencies that are not rational multiples of each other. Our architecture relies on the non-repeating phase relations among these incommensurable analog oscillators to drive the search for optimal solutions, rather than making use of external sources of noise or relying on random fluctuations. I show that the architecture naturally reproduces the dynamics of stochastic local search (SLS) algorithms. These algorithms are typically incomplete, i.e, they can not prove that a solution does not exist for unsatisfiable problems. Complete algorithms can also be mapped to the proposed architecture, albeit not as naturally as SLS algorithms, as I show when discussing the Boolean satisfiability problem. This chapter presents a physical architecture for solving CSPs, the *inferenceEngine* architecture, together with massively parallel reformulations of well-established CSP algorithms so that they can be efficiently instantiated on the proposed architecture. I present results from an implementation of this architecture on a prototype VLSI chip. These results expose a surprising relation between the dynamics of coupled multi-stable oscillators and the search for CSP solutions and highlight a novel mode of distributed, parallel, mixed analog/digital computation that can form the basis of various hardware/physical systems for solving CSPs.

7.1. Description of the InferenceEngine Architecture

The proposed event-based architecture for solving CSPs employs a network of nodes which communicate via digital events. A node is shown schematically in Fig. 7.1a. Each node has N externally accessible input ports, one internal input port, M output ports, and one dummy output port. The analog oscillator in the node generates a continuous stream of digital events which are sent to the node's internal port: $in.0$. The digital logic in the node has an internal state s which can take one of Q possible values. On the arrival of an event on any of the input ports, the node's digital logic evaluates the index of the output port to which it should send the event based on the index of the triggered input port and on the current state of the digital logic; it updates its internal state; and it transmits the event via the output port selected (see Fig. 7.1b). Selection of the 'dummy' output port $out.0$ is equivalent to suppressing the event. The digital logic is fully described by the event routing function g and the state update function f which are both deterministic. Given their analog nature, the natural frequencies of the oscillators in the different nodes are not rational multiples of each other. Due to fabrication-induced mismatch, different oscillators realized on a VLSI chip will have incommensurable natural frequencies. It is important that coupling between these physical oscillators be kept at a minimum so as to minimize the chance of phase locking.

For solving CSPs, a subset of the nodes in the network will represent the actual problem variables while others will represent helper variables that encode other problem-relevant quantities (for example, whether a constraint is satisfied or not). The value of a variable/node at a point in time is the index of the output port on which the node emitted its last event. Thus, a variable/node with M output ports can have M possible values. The output port of one node can connect to the input ports of one or more nodes and one input port can receive events from multiple output ports. One output port can not be connected to multiple input ports on the same node. The following sections describe how to connect nodes/variables together and how to define the nodes/variables behavior (the f and g functions) so as to solve a number of hard CSPs. The procedure to map a CSP to this distributed architecture depends on the type of the CSP but in general, the mapping is done so that the distributed and parallel dynamics of the network of nodes tries to put the problem variables/nodes in a state where their outputs satisfy all the

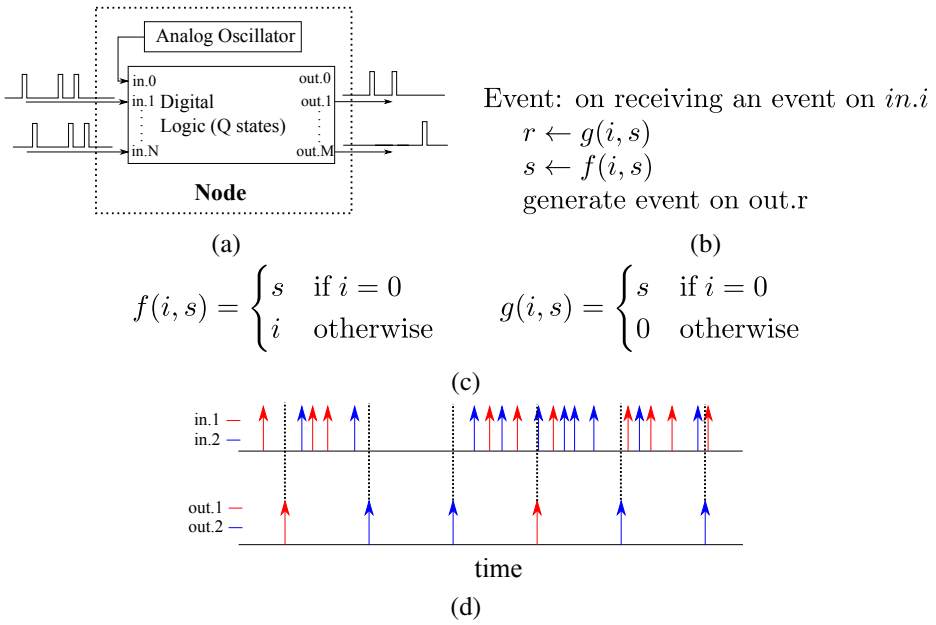


Figure 7.1: Building blocks of the *inferenceEngine* architecture. (a) General form of the computational unit in our architecture. This computational unit, or node, is composed of asynchronous state-holding digital logic, and an analog oscillator that generates a stream of events. The digital logic is event-driven and changes its internal state in response to events on its input ports $in.0$ to $in.N$. The node can generate an event on one of the output ports in response to input events. (b) Formal description of a node. On an input event on port $in.i$, the digital logic evaluates the index of the output port r, updates its state s, and generates an event on output port $out.r$ in that order according to the functions f and g. Events from the internal oscillator always arrive on the $in.0$ port. An output event on the $out.0$ port is discarded, i.e, the node does not generate an output event if $r = 0$. (c) Definition of the f and g functions for an example binary node with two internal states, two input ports, and two output ports ($N = M = Q = 2$). (d) Simulation of the example node showing its input and output event streams. The node generate an output event for each event from the periodic internal oscillator. The output events reflect the identity of the last input event the node received.

constraints

Figures 7.1c and 7.1d show the definition and illustrate the behavior of an example node which has two input ports, two output ports, and two possible internal states ($N = M = Q = 2$). The state of the example node/variable is the index of the last external event it received and the node/variable advertises its state by generating an event on one of the output ports when it receives an event on the internal port $in.0$ as shown in Fig. 7.1c (I refer to this as 'updating'). Assume this example node is the target node receiving events from multiple sources nodes. Since it is only the last received event that determines the value advertised by an updating node, the phase relations between the analog oscillators in the network determine which of the source nodes generates the decisive event that determines the event generated by the target node. This would be the source node that updated just before the target node updates. The phase relations are continuously changing in an aperiodic manner since the oscillation frequencies are incommensurable. The shifting phase relations thus continuously change which source node

manages to influence the output events of the target node.

For the node described in Fig. 7.1c, assume N_1 nodes with frequencies $f_1^1, f_2^1, \ldots, f_{N_1}^1$ are sending events to its $in.1$ port and N_2 nodes with frequencies $f_1^2, f_2^2, \ldots, f_{N_2}^2$ are sending events to its $in.2$ port, the fraction of 1 events generated by the target node converges to $\sum_{i=1}^{N_1} f_i^1 / (\sum_{i=1}^{N_1} f_i^1 + \sum_{j=1}^{N_2} f_j^2)$ if observed for a long enough time. Assuming the differences in oscillator frequencies of the source nodes are small, the latter expression can be written as $\frac{N_1}{N_1+N_2}$. Thus, the more nodes that try to force a target node to a particular value, the more likely the target node is to output that value, yet there is always a chance that even a single source node that is in conflict with the majority will update just before the target node updates, thereby causing the target node to go against the majority influence. As I will show in the next sections, this behavior can be exploited to allow the network to escape from local minima where flipping a single variable/node may increase the number of violated constraints. However, a node will never take a value that is in conflict with all incoming influences which is why the globally optimal state is stable. I show that this mostly greedy, but sometimes exploratory, behavior can be exploited to efficiently solve a variety of hard CSPs.

In contrast to simulated annealing, there is no temperature parameter that balances the greedy versus the exploratory aspect of the network behavior. The exploration of the solution space is not noise-driven but, rather, it is driven by the continuously changing, non-repeating order of event generation. This has the advantage that no cooling schedule is needed and there is no artificial separation between an exploratory (high temperature) phase and a greedy (low temperature) phase. This enables the problem to be dynamically changed and new constraints added without having to restart any cooling schedule. Due to the incommensurable oscillation frequencies, the network trajectory is aperiodic, so the system does not get stuck in loops. One disadvantage, however, of using deterministic incommensurable oscillators to drive the search is that I can not derive probabilistic convergence results as in simulated annealing.

7.2. Boolean Satisfiability Problems

Let $\mathbf{X} = \{x_1, \ldots, x_N\}$ be a set of Boolean variables. A literal is either a variable or its negation. The solution to a Boolean satisfiability or K-SAT problem is the variable assignment that satisfies the logical expression involving the variables of $\mathbf{X}$:

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m = \text{True}, \tag{7.1}$$

where the clause c_i is the disjunction of K literals. K-SAT for $K \geq 3$ is NP-complete (Sipser, 1996).

7.2.1. The ProbSAT Algorithm

The implementation of the probSAT algorithm on the *inferenceEngine* architecture and the analysis of its performance was done by Lorenz Muller.

A highly efficient algorithm for solving random SAT problems is the probSAT algorithm (Balint & Schöning, 2012), winner of the parallel random track of the 2014 satcompetition (Belov et al., 2014). probSAT iteratively modifies a variable assignment by choosing a random unfulfilled clause c_u and changing the assignment of ('flipping') a random variable x_f in c_u, thereby fulfilling c_u. The choice x_f is governed by a heuristic function $f(m, b)$, where m (the 'make' heuristic) is the number of clauses that are newly fulfilled when x_f is flipped and b (the 'break'

heuristic) is the number of clauses that are newly unfulfilled when x_f is flipped. The heuristic function is renormalized into a probability over the available choices and x_f is chosen according to these probabilities. The heuristic function $f(m, b)$ can take several different forms. In our benchmarks, we use the particularly effective 'exponential' form:

$$f(m, b) = \frac{x^m}{y^b}, \tag{7.2}$$

where x and y are parameters. If no solution is found after n_{max} flips, all variables are set to new random values, i.e, the algorithm is restarted.

We map the probSAT algorithm with only the 'break' heuristic to our architecture by using two types of nodes: nodes representing variables and nodes representing constraints/clauses. Each variable node has two states. It updates and advertises its state (by generating an event on one of its two output ports) whenever it receives an event from a clause node. Additionally, it advertises its state whenever it receives an event from the internal oscillator.

Figure 7.2 shows a sample 3-SAT network. When the clause node receives a break event (event arriving on one of its break input ports, one corresponding to each variable), it increments the corresponding break counter. On events from the internal oscillator, a clause node evaluates what state the connected variables have last advertised. If there is no variable in a fulfilling state, the clause node sends an event to flip the variable with the smallest associated 'break' count and sends a 'break' event to every clause node in which this variable appears with the opposite polarity to indicate that the flipped variable is the only variable keeping the constraint fulfilled. If there is exactly one fulfilling variable, the clause node only sends a 'break' event for that variable; every r^{th} 'break' event is skipped and not sent where r is a fixed parameter. If there is more than one fulfilling variable, the clause node does not send out any events. The break counters are reset after each event from the internal oscillator.

An unfulfilled clause node thus always chooses to flip the variable with minimal break count (with ties resolved according to a fixed variable ordering). The skipping of some break events implements a 'softening' of this hard minimum function. This flip heuristic is deterministic and simpler than the heuristic employed by standard probSAT.

In ref. (Mostafa et al., 2015b), Lorenz Muller used a network of oscillatory nodes such as the one shown in Fig. 7.2 to solve a number of 3-, 4-, and 5-SAT problems and showed that in order to match the time to solution of a current-generation standard CPU, the oscillation frequencies of the nodes in the probSAT network should be around 2 MHz. A potential bottleneck in the *inferenceEngine* architecture is the event routing fabric. We can calculate the approximate routing rate necessary to match the standard implementation of probSAT by dividing the number of events to solution by the median runtime of standard probSAT on a CPU. This yields a required combined event routing rate of around 100 GEv per second. Note that a single event generated by a node is typically dispatched to multiple nodes. An event generated by a node is thus typically split into multiple events, each targeting a single node. Each of these "split" events is separately counted when calculating the required event rate. A 100 GEv per second rate can be achieved if the highly local nature of node communication (which reflects the local nature of the constraint graph) is exploited by a parallel event routing scheme, and if the event routing scheme supports multi-casting that allows a single source event to be efficiently routed to multiple destinations.

7.2.2. Mapping a Complete SAT Solver to a Network of Nodes:

The *inferenceEngine* architecture is well suited to implementing stochastic local search (SLS) algorithms such as probSAT as it can exploit the non-repeating phase relations among the in-

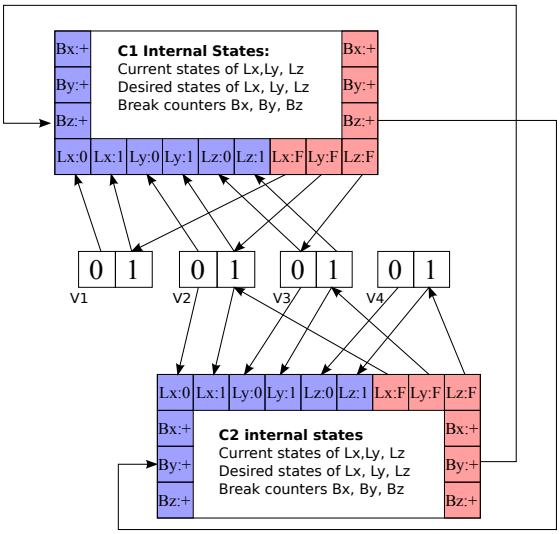


Figure 7.2: Sample network implementing probSAT. Network corresponding to the example SAT problem $C1 \wedge C2$ where $C1 = (V1 \vee V2 \vee \neg V3)$ and $C2 = (V2 \vee V3 \vee V4)$. For the constraints $C1$ and $C2$, the squares at the edge of the box indicate input ports (blue) and output ports (red). Events are routed along the arrows. Each unfulfilled constraint node periodically choose a variable in its domain to flip. The chosen variable is the one with the lowest break count, i.e, the variable that will cause the smallest number of other clauses/constraints to be unfulfilled when flipped. A constraint node updates its break counters based on the events it receives from other constraint nodes that have one or more variables with opposite polarity in common.

commensurable oscillators as a source of non-repeating noise that drives the search . Local search algorithms are typically incomplete, i.e, they can not prove that a solution does not exist for unsatisfiable formulae. I now describe how a complete SAT algorithm can be mapped to the *inferenceEngine* architecture. Complete algorithms for solving Boolean satisfiability problems typically make use of the DPLL procedure shown in Algorithm 1. This is a recursive sequential procedure where at each recursion level, a decision variable is chosen and assigned a value. The algorithm backtracks to an earlier recursion level if a violation is detected. At the beginning of each recursion level, Boolean constraint propagation (BCP) first identifies all variables whose values are implied by the current partial assignment and adds these variables to the current partial assignment. For example, the only unassigned variable in an unsatisfied clause is assigned a value by BCP to satisfy the clause (unit clause rule). The DPLL algorithm forms the core of many modern SAT solvers. The algorithm is made more efficient through the use of advanced heuristics to pick the literal at each recursion level (Moskewicz et al., 2001), and most crucially through the use of conflict driven clause learning that augments the original clauses with new clauses when a conflict is detected (Silva & Sakallah, 1997). This enables later conflicts to be detected earlier in the search tree and thus leads to more efficient pruning of the tree.

I use an algorithm that is custom-tailored to the architecture and that combines an SLS-like search inspired by probSAT with a systematic pruning of the SAT solution tree inspired by DPLL. The clause and variable nodes are shown in Fig 7.3. Each SAT variable can have four values: 0, 1, 0F, or 1F. The latter two values indicate that the variable value is frozen at 0 or 1 respectively, i.e, it has been fixed either through implication from other frozen variables through BCP, or by being chosen as the decision variable at some DPLL recursion level. The Boolean internal state freeze_type distinguishes between these two cases. Each variable ad-

Algorithm 1 DPLL(F,P)

Input: Formula F, partial assignment P
Output: Satisfying assignment or False
$P_{bcp} \leftarrow P \cup \textbf{boolean\_constraint\_propagation}(F, P)$
if P_{bcp} is a complete consistent assignment **then**
    return P_{bcp}
else if contradiction(F,P_{bcp}) **then**
    return False
else
    $l \leftarrow$ **select_literal**(F,P_{bcp})
    // short-circuit 'or' operator. Returns non-False argument without casting to Boolean
    return DPLL (F,$P_{bcp} \cup \{l = 1\}$) or **DPLL** (F,$P_{bcp} \cup \{l = 0\}$)
end if

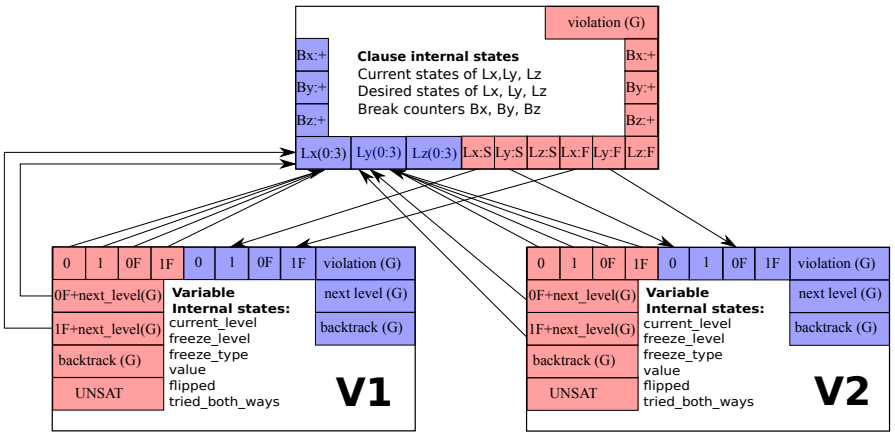


Figure 7.3: Sample network implementing hybrid probSAT/DPLL. The variable and clause nodes implementing the hybrid DPLL/probSAT scheme for the clause $V1 \vee \neg V2$. The squares at the edge of the boxes indicate input ports (blue) and output ports (red). Events are routed along the arrows. Events from output ports with the (G) postfix are routed to all input ports with the same name and the (G) postfix (arrows not shown) of all variables.

vertises its state each cycle by generating an event on one of the output ports: 0, 1, 0F, or 1F. The current_level internal state is an integer denoting the current DPLL recursion level while freeze_level indicates for frozen variables the recursion level at which they have been frozen.

Initially, all variables are either 0 or 1, current_level and freeze_level are both zero, and flipped and tried_both_ways are both false. probSAT proceeds as described before. When a variable's advertised state at the end of one cycle is different from its advertised state in the previous cycle (because it has been flipped by a clause), it sets its internal variable flipped to true. At the end of the next cycle and if flipped is still true, the variable freezes (takes the value 0F or 1F), sets its freeze_level to current_level+1 and freeze_type is set to denote that the variable has been frozen because it is the decision variable for a recursion level. The variable generates an event on either "0f+next_level" or " 1f+next_level". This event advertises its frozen state and is routed to the "next_level" input port on all variables which causes all variables to increment their current_level internal counter and reset their flipped internal state to false. tried_both_ways is set to false in the frozen variable. In subsequent cycles, the frozen

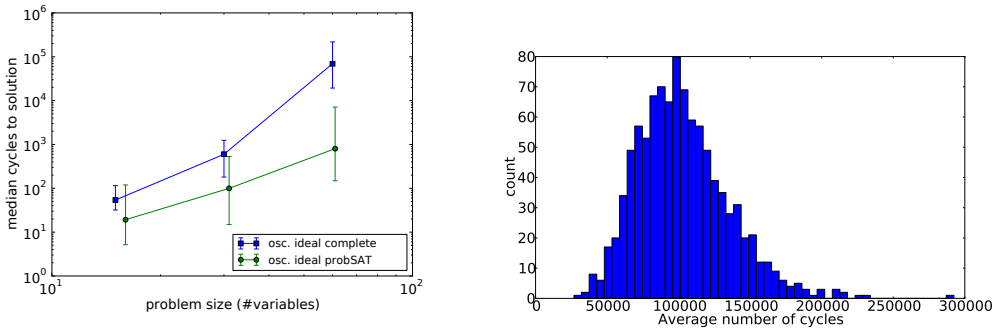
variable advertises its state through the "0F" or "1F" output ports. Unfrozen variables continue to be flipped through the probSAT dynamics and gradually more of them become frozen and the recursion level (which is stored in the internal counter current_level that is equal in all variables) increases.

Each clause has up-to-date information about the state of the variables in its domain. Unsatisfied clauses flip unfrozen variables in their domains according to the probSAT algorithm. When an unsatisfied clause detects that only one variable in its domain is unfrozen, it sends an event to freeze this variable at a satisfying assignment. This frozen variable sets its freeze_level to the current recursion level (which is available in its current_level counter) and sets its freeze_type to denote that it has been frozen due to the unit clause rule. At some point, an unsatisfied clause node may detect that all its variables are frozen, i.e, a contradiction has been reached. It then generates an event on its "violation" output port which is routed to all variables. All variables frozen through unit clause propagation whose freeze_level is equal to current_level become unfrozen, i.e, their values become either 0 or 1. There is bound to be only one variable frozen as a decision variable whose freeze_level is equal to current_level. If tried_both_ways in this variable is false, this variable simply flips its frozen value and sets tried_both_ways to true. Otherwise, both values have been tried and the variable takes an unfrozen value and generates an event on the "backtrack" output port. All variables receive the backtrack event, and in response decrement the current_level counter. The backtrack event is then treated as a violation event which will cause variables frozen through unit clause propagation at the current level to unfreeze, and a decision variable to either flip or generate further backtrack events. If a frozen decision variable whose freeze_level is 1 (the first decision variable) and which has been tried both ways receives a "violation" or a "backtrack" event when current_level is 1, then it generates an event on the UNSAT port to signal that the problem is unsatisfiable.

This scheme for freezing and unfreezing variables is analogous to the DPLL procedure in Algorithm 1 with one important difference: BCP or unit clause propagation is always active and proceeds in parallel with the mechanism for the selection and freezing of new decision variables. This may cause a violation caused by the freezing of a decision variable to be detected at deeper/later recursion levels but the violated clause will keep generating violation events until the network has backtracked to the problematic decision variable assignment. A significant part of the execution time of many complete SAT solvers is spent in the unit clause propagation phase. Unit propagation in the described scheme is done in a parallel fashion across all clauses and is thus quite fast. In parallel to the DPLL mechanism, the probSAT mechanism is searching for satisfying assignments in the unfrozen part of the network.

The network of nodes implementing the complete algorithm is guaranteed to either find a solution, or to generate an event indicating that the problem is unsatisfiable. One disadvantage of this complete algorithm is that it requires certain events to be globally routed to all nodes, thus degrading the efficiency of any parallel event routing scheme that seeks to exploit the local nature of inter-node communication. These global events are a reflection of the typically centralized and sequential nature of complete algorithms.

The network of nodes implementing the complete algorithm is not as efficient as the network implementing the probSAT algorithm when solving satisfiable K-SAT instances. This is illustrated in Fig. 7.4a where the algorithm takes significantly more cycles to find a solution. Though not as efficient as the probSAT network, the network implementing the complete algorithm can detect that a problem is unsatisfiable as shown in Fig. 7.4b.



(a) (b)

Figure 7.4: Performance of the network implementing the complete SAT algorithm. (a) Median of the average number of oscillation cycles to solution (averaged over the nodes in the network) for the ideal oscillatory network implementing probSAT and for the ideal oscillatory network implementing the complete algorithm when solving random 4-SAT instances with $\alpha_4 \in [9.4, 9.45]$. Error bars show 1st and 3rd quartiles. The network implementing the complete algorithm is slower than the one implementing probSAT when searching for solutions to satisfiable instances (b) Histogram of the average number of cycles (averaged over the nodes in the network) taken by the network implementing the complete algorithm to signal that a 3-SAT instance is unsatisfiable. Network was run once per instance on 1000 random 3-SAT instances with 100 variables and 430 clauses each.

7.3. Graph Coloring Problems

The graph coloring network was developed in collaboration with Lorenz Muller and the simulations were run by Lorenz Muller.

A k-coloring for an undirected graph G with vertices $V(G)$ and a set of edges $E(G)$ is a map $\phi : V(G) \to \{1, 2, \ldots, k\}$. In the graph coloring problem, the goal is to find a proper k-coloring ϕ_0 of G where $\phi_0(x) \neq \phi_0(y)$ for all $\{x, y\} \in E(G)$.

To solve a k-coloring problem, we map each vertex in the graph to a network node with k input ports and k output ports as shown in Fig. 7.5. Whenever the internal oscillator in a node/vertex generates an event, the node advertises its color by generating an event on one of the k output ports. Events from a node/vertex are routed to all its neighbors in the graph. Each node maintains k counters that count how many of its neighbors have a particular color. These counters are incremented when a node receives events from its neighbors. At an internal oscillator event, if the counter corresponding to the current node color is non-zero (one of the neighbors has the same color), the node chooses a different color. If the internal Boolean variable, 'heuristic', is true, the node chooses the color with the fewest conflicts (smallest neighbor count). If 'heuristic' is false, the node chooses the next color in a fixed arbitrary ordering of colors. The node then resets the k counters, flips the 'heuristic' binary variable and generates an event to advertise its color. A min conflict heuristic thus alternates with a heuristic free scheme to update a conflicting node each cycle.

We assessed the performance of this algorithm on several k-coloring problems of intermediate difficulty (see Table 7.1) taken from (Ruiz & Romay, 2011) in which a different massively parallel coloring algorithm ('gravitational swarm intelligence' (GSI)) was assessed. We cannot attempt state-of-the-art sized problems since the software simulation of a large network takes an infeasibly long time. In terms of average numbers of oscillation cycles to solution the network

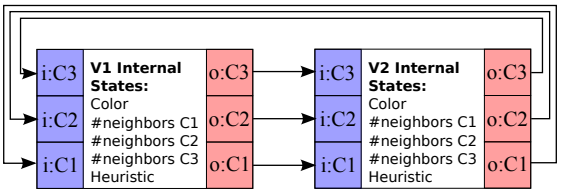


Figure 7.5: Network solving graph coloring. Network corresponding to the 3-coloring of the graph $V = \{V1, V2\}$, $E = \{(V1, V2)\}$. The squares at the edge of the box indicate input ports (blue) and output ports (red). Events are routed along the arrows. When two connected nodes represent the same color, one of them will change its color on its next internal oscillator event.

Graph	#vertices	#edges	Density	K	#GSI	#network
myciel7	191	2360	0.13	8	302	**145**
myciel6	95	755	0.17	7	92	**31**
myciel5	47	236	0.21	6	97	**19**
myciel4	23	71	0.28	5	25	**3**
myciel3	11	20	0.36	4	21	**2**
david	87	986	0.21	11	208	**95**
anna	138	812	0.21	11	300	**8**
huck	74	662	0.22	11	84	**8**
jean	80	508	0.16	10	165	**16**
queen 5x5	25	160	0.53	5	**302**	N/A
1_fullins_3	30	100	0.23	4	37	**11**
1_fullins_4	93	593	0.14	5	**76**	366
1_fullins_5	282	3247	0.08	6	**222**	1593
2_fullins_3	52	201	0.15	5	67	**47**
2_fullins_4	212	1621	0.07	6	176	**120**
miles_250	128	387	0.04	8	**317**	2021

Table 7.1: Performance of the graph coloring network. Number of cycles to convergence on common k-coloring benchmarks (Ruiz & Romay, 2011) of our network and a massively parallel algorithm (Ruiz & Romay, 2011). Each number in the network column is an average of 4 runs with redrawn oscillator frequencies; one run for the queens graph did not converge in 10^5 cycles (the other runs averaged 530 steps to convergence).

compares favorably to GSI (Ruiz & Romay, 2011).

7.4. Implementing Analog Costs and the Traveling Salesman Problem

The problems considered so far, 3-SAT and graph coloring problems, have hard constraints that should all be satisfied. The *inferenceEngine* architecture can also handle CSPs with weighted constraints by exploiting the relations between the frequencies of the analog oscillators. I illustrate this approach using the traveling salesman problem (TSP). A TSP with N cities is defined

by the $N \times N$ matrix $\mathbf{D}$ where d_{ij} is the distance from city i to city j. The goal is to find a minimum length closed tour of all the cities. We can not require a distributed architecture like the one presented to stabilize at the optimal solution as there is no general way, short of using long-term memory resources for storing previously visited tours, to verify that a tour is optimal. Our goal is thus to map a TSP to a network of nodes that continuously explores all possible valid tours but that has a higher chance of visiting tours with smaller distances.

I only consider symmetric TSPs where $d_{ij} = d_{ji}$. I associate nodes in the network with directed edges in the TSP tour. An edge from city i to city j is represented by node (i, j). Each edge node has 3 possible internal states, 3 input ports and 1 output port and an event from edge node i, j means that this edge is present in the tour. Edge nodes are described by the state update function f_{edge} and spike routing function g_{edge} (see Fig. 7.1b) given by:

$$f_{edge}(i,s) = \begin{cases} 3 & \text{if } (i = 4 \; or \; (i = 2 \; and \; s = 2)) \\ 2 & \text{if } i = 3 \\ 1 & \text{if } (i = 1 \; or \; (i = 0 \; and \; s = 3)) \\ s & \text{otherwise} \end{cases} \qquad g_{edge}(i,s) = \begin{cases} 1 & \text{if } (i = 0 \; and \; s = 3) \\ 0 & \text{otherwise} \end{cases}$$

The node will only generate an event when it is in state 3 (activated state) and when the internal oscillator generates an event. It then goes to state 1 (deactivated state). The node will go to state 3 if it receives an event on input port 4 or if it is in state 2 and receives an event on input port 2. I first consider a six cities symmetric TSP which can be mapped to the network shown in Fig. 7.6a. Row i contains edge nodes for edges originating from city i and column j edge nodes for edges that terminate at city $j+1$. Events from edge node (i, j) are routed to port 1 of all edge nodes in the same row and column, and to port 2 on all edge nodes in row j. Assume initially all edge nodes are in state 2 except the nodes in row 1 which are in state 3 (activated). One of the nodes in row 1, for example $(1, 3)$, will generate an event first and deactivate all edges nodes originating from city 1 or terminating on city 3 by putting these edge nodes in state 1. The $1, 3$ event activates all edge nodes originating from city 3 (edge nodes in row 3) by putting them in state 3 (Fig. 7.6a). Assume $(3, 2)$ generates an event first in row 3 thereby disabling all edges originating from city 3 or terminating on city 2. The event from $(3, 2)$ switches edge nodes in row 2 that were in state 2 to state 3 (Fig. 7.6b). One of the activated edge nodes in row 2 will now generate an event and the sequence continues (Fig. 7.6c). Through the events of the edge nodes (exactly 5 nodes will generate an event), we obtain a valid tour, for example: $(1, 3)$; $(3, 2)$; $(2, 4)$; $(4, 6)$; $(6, 5)$. After these 5 events, all edge nodes are in state 1 (deactivated).

The chance for an edge node (i, j) that is at state 3 to "win the race" and send an event first when row i is activated depends on its frequency $f_{i,j}$ and the frequencies of the other active nodes in the row. As $f_{i,j}$ increases, the likelihood for edge node (i, j) to generate an event first and become part of the tour when its row is activated increases. To increase the "probability" of obtaining a short tour, we set $f_{i,j}$ to:

$$f_{i,j} = \frac{K}{d_{i,j}} + \eta_{i,j}$$

where K is a positive scaling constants and $\eta_{i,j}$ is a small perturbation to keep the frequencies incommensurable. The node of a shorter edge has a higher frequency which makes the edge more likely to appear in the tour.

The 'tour completion' node in Fig. 7.6a is responsible for resetting the network at the end of a valid tour. It has 2 internal states, 1 input port, and 1 output port and is defined as:

$$f_{tc}(i,s) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = 0 \end{cases} \qquad g_{tc}(i,s) = \begin{cases} 1 & \text{if } (i = 0 \; and \; s = 2) \\ 0 & \text{otherwise} \end{cases}$$

The frequency of the 'tour completion' node is chosen to be slightly less than that of the edge node having the smallest frequency and it receives events from all edge nodes. If the 'tour completion' node does not receive any event within one oscillation cycle (all edge nodes are inactive), it generates an event that goes to port 4 of the edge nodes in row 1 and port 3 of all other edge nodes thereby resetting the network and starting the tour generation process.

Figure 7.6d shows the frequency of occurrence of the tours generated by a network encoding a six city TSP. There are 120 tours as the first city in the tour is always city 1. Tours having the same distance do not occur equally often and some tours with longer distances occur more often than tours with shorter distances. In general, however, shorter tours tend to occur more often.

For large problems, the performance of the TSP network falls far short of state of the art TSP algorithms. TSPs have a global constraint which is the requirement that a tour is valid. The *inferenceEngine* architecture is ill-suited to handling such a global constraint as different parts of the network can no longer operate in parallel to optimize the local constraints. As the size of the TSP increases, the fraction of valid tours decreases exponentially which makes it imperative that this global constraint be strictly enforced, otherwise the obtained tours will mostly be invalid. The presented scheme for solving TSPs is pseudo-sequential, which is counter to the distributed nature of the architecture, where edges are added one by one to keep the global tour validity constraint satisfied at all times. Even though the TSP implementation described is inefficient, it shows that analog costs can be implemented through the appropriate choice of node frequencies and that the deterministic network can exhibit a sampling-like behavior. In implementations where individual node frequencies are not controllable, the same effect can be achieved by choosing nodes having the appropriate frequency relations from a large pool of mismatched nodes.

7.5. Prototype Implementation of the InferenceEngine Architecture

The prototype VLSI chip that implements a simple version of the *inferenceEngine* architecture is composed of a 2D array of binary nodes that communicate using events. The problem of transmission and routing of asynchronous events has been thoroughly investigated in the neuromorphic engineering literature (Deiss et al., 1998; Boahen, 2000). An elegant solution uses a communication protocol based on the Address-Event Representation (AER). When a node generates an event on one of its output ports, it executes a handshake protocol with the 'output AER interface'. The 'output AER interface' encodes the address of the output port on which the event was generated and transmits the address off-chip using an output bus that has $log_2(K_{out})$ lines. K_{out} is the number of possible event sources (the output ports of all the nodes). The array has K_{in} possible event targets (the input ports of the nodes), if an event is to be sent to one of these targets, the target address is sent to the 'input AER interface' on a bus that has $log_2(K_{in})$ lines. The 'input AER interface' decodes the address and sends an event to the target element by simultaneously activating the correct row and column in the array.

The 2D array on the chip comprises 64*32 binary nodes/variables, i.e, nodes/variables with two output ports. The chip can be configured so that 2,3, or 4 adjacent variables are merged together to realize 4-, 6-, or 8-valued variables respectively. An n-valued variable ($n \in \{2, 4, 6, 8\}$) has n output ports and n possible internal states and has $2^n - 1$ input ports. Physically, a variable has n digital input lines on which it receives a binary word encoding the index of the input port receiving the event. An off-chip event router implemented on a field programmable gate array (FPGA) communicates with the output and input AER interfaces to route events from

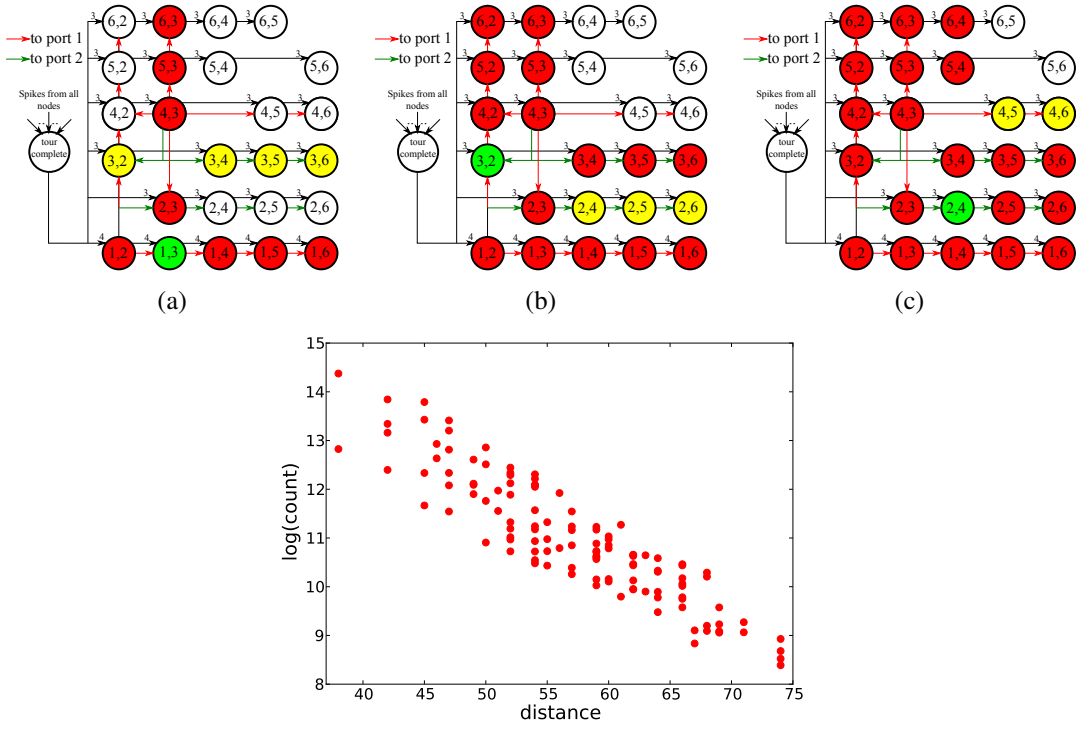


Figure 7.6: (a, b, c) TSP with 6 cities. Red and green arrows indicate inputs to ports 1 and 2 respectively. Numbers next to black arrows denote target input ports. Only the routing of the events of edge nodes $(1,2)$ and $(4,3)$ and of the 'tour completion' node are shown. A green node is a node that has just generated an event, red nodes are in state 1 (inactive), yellow nodes are in state 3 (active). a, b, c show the network state after each event for 3 successive events. (d) Frequency of occurrence of each tour in $1.5{*}10^7$ tours generated by a network implementing a six cities TSP problem as a function of the tour distance.

nodes/variables output ports to input ports according to a programmable routing table.

When an n-valued node/variable receives an event on port i, the 1s in the binary representation of i denote the allowable internal states that the variable can take. The variable has n possible internal states and an event on one of the $2^n - 1$ input ports can thus decide which non-empty subset of these states are allowed. If multiple states are allowed, the variable stays at its current state if the current state is one of the allowed states, otherwise it goes to the lowest index allowed state. Let $i(p)$ be the p^{th} bit of i where indexing starts at 1, the state update function f is thus:

$$f_{HW}(i,s) = \begin{cases} s & \text{if} \quad i(s) = 1 \;\; or \;\; i = 0 \\ p & \text{if} \quad (i(s) = 0 \;\; and \;\; i(p) = 1 \;\; and \;\; i(j) = 0) \;\; \text{for} \;\; j = 1, 2, \ldots, p-1 \end{cases} \tag{7.3}$$

The node/variable generates an event only when it receives an event on port 0. The event is generated on the port corresponding to the currents state. The event routing function g is:

$$g_{HW}(i,s) = \begin{cases} s & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

The analog oscillator in each variable is realized using an integrate and fire neuron (Indiveri

et al., 2006) receiving constant current injection. As shown in Fig. 7.7c, the oscillator frequencies are significantly different due to transistor mismatch. Since the oscillation frequencies are real numbers drawn from a probability distribution arising from the variability inherent in the fabrication process, it is impossible for an oscillator to have a natural frequency that is a rational multiple of another's. Coupling between the oscillators, however, could cause two oscillators with nearby frequencies to lock and effectively have the same frequency. Synchrony between the oscillators could be characterized using the mean phase coherence(MPC) measure, which for two waveforms with instantaneous phases $\phi_1(t)$ and $\phi_2(t)$ is defined as:

$$MPC(\phi_1(t), \phi_2(t)) = \left| \frac{1}{T} \int_T e^{j(\phi_1(t)-\phi_2(t))} dt \right| \tag{7.5}$$

I turned on all 2048 oscillators and assumed the phase changes linearly from 0 to 2π between successive events from an oscillator. I calculated the MPC for each pair of oscillators and the distribution of MPC values is shown in Fig. 7.7d (red line). In Eq. 7.5, I used a time discretization of $0.1\,ms$ and an integration time of $18\,s$. I randomly generated 2048 double-precision frequencies from a Gaussian distribution having the same mean and variance as the frequency distribution on the chip and generated an artificial constant-frequency waveform for each of these artificial frequencies. Using the same time discretization and total integration time in Eq. 7.5, I evaluated the distribution of MPC values for all possible pairs of the artificial constant-frequency waveforms. Ideally, all these MPC values would be zero, but as shown in Fig. 7.7d, due to discretization and finite integration time, these uncoupled artificial waveforms have non-zero MPC values. The oscillators on the chip are more synchronized compared to the ideal case (uncoupled oscillators) as evidenced by the heavier tail of their MPC distribution. Oscillator coupling is a potentially serious problems as it compromises the exploration of all possible phase relations which is key to the exploration of the solution space. Most pairs of on-chip oscillators, however, exhibit very low MPC values that are on par with the MPC values for uncoupled oscillators. Maximum MPC value was 0.34 so no phase locking was observed.

7.5.1. Solving 3-SAT Problems on the Hardware Prototype

Using the node logic on the prototype chip, I implemented a simple 3-SAT algorithm. This 3-SAT algorithm is based on the analogy established by Lorenz Muller between the chip dynamics and the stochastic local search algorithm from ref. Papadimitriou (1991). Each problem variable is represented by one binary node and each 3-SAT constraint/clause is represented by a 4-valued node. An event from the 1 port or the 2 port of a binary variable/node denotes that the variable value is 0 or 1 respectively. The constraint/clause node is in state 4 if the constraint is fulfilled, otherwise its state (1 or 2 or 3) denotes which literal in the constraint last emitted an event. Consider a 3-SAT constraint $C1 = (L1 \vee L2 \vee \neg L3)$. Events from port 2 of variables/nodes $L1$ and $L2$ and events from port 1 of $L3$ should put the $C1$ node at state 4 (constraint fulfilled). A complementary event, i.e, an event that does not cause the constraint to be fulfilled (for example a 2 event from $L3$) should do nothing if the constraint is fulfilled as we assume one, or both, of the other two variables fulfill the constraint. However, if the constraint is not fulfilled, a complementary event from the k^{th} variable in the constraint should put the constraint node in state k.

When the constraint node advertises its state by an event, events from ports 1, 2, or 3 should set the first, second, or third variable respectively to a constraint-fulfilling state. In the scheme described so far, when a constraint node is fulfilled (in state 4), events from the variables will never move it away from the fulfilled state. To address this, whenever the constraint node generates an event on port 4, this event is routed back to the constraint node and moves it to an

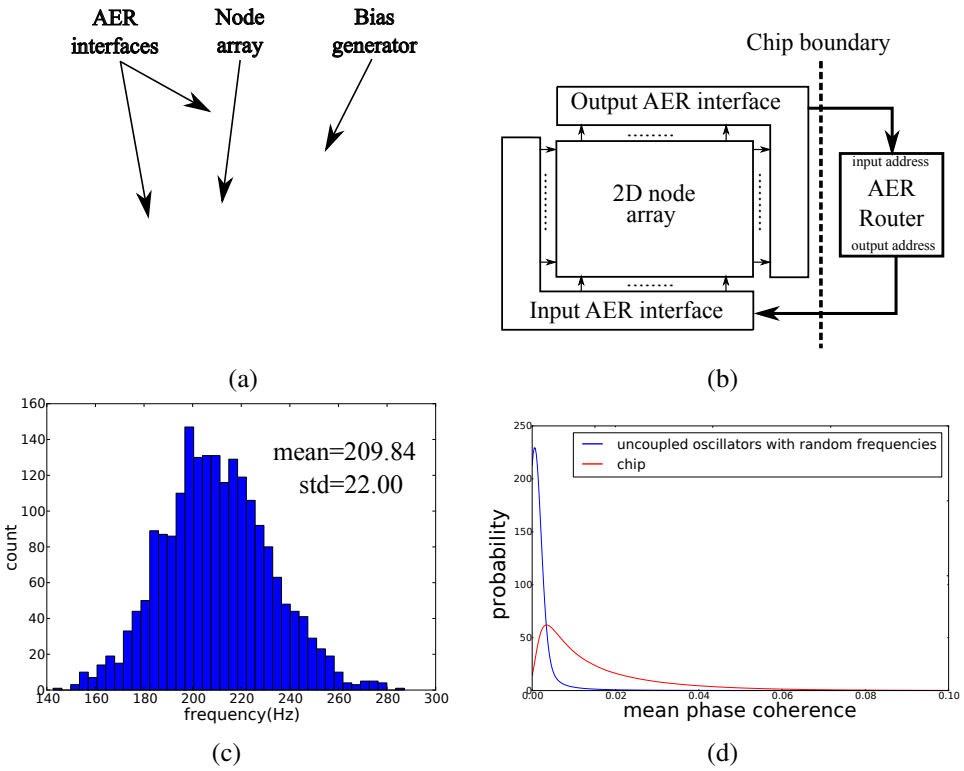
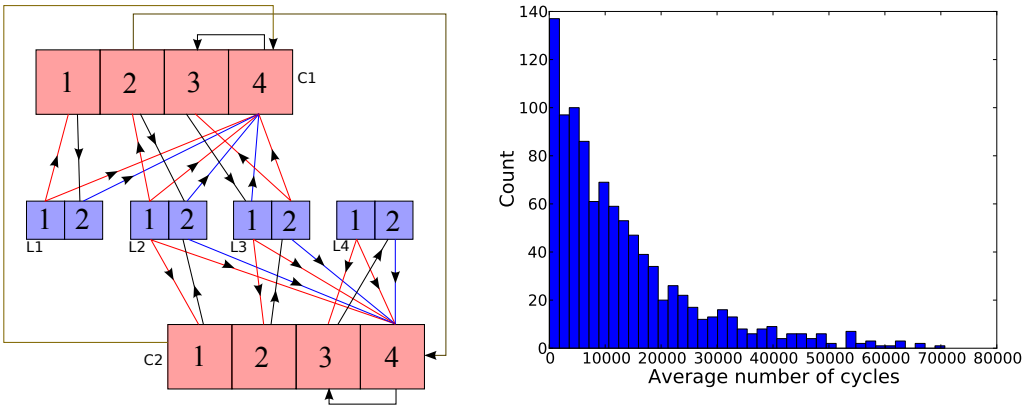


Figure 7.7: (a) Layout of the minimum size (2mm*3mm) prototype chip fabricated using a 180 nm CMOS process that implements the *inferenceEngine* architecture. The 64*32 node array in the middle is surrounded on three sides by the digital asynchronous AER interfaces. An externally programmable bias generation block generates the analog biases needed by the analog oscillators.(b) An off-chip event router implemented on an FPGA communicates with the chip AER interfaces to route events from output ports to input ports. (c) Frequency distribution of the 2048 on-chip analog oscillators for the bias conditions used in the experiments in this chapter. (d) Normalized distribution of MPC values for all pairs of physical oscillators on the chip, and for all pairs of 2048 artificially generated constant-frequency waveforms.

arbitrary unfulfilled state (I arbitrarily choose state 3). Thus, within each oscillation cycle of the constraint node, the node has to receive a constraint-fulfilling event from one of its variables in order to go to state 4 and in order not to generate an event at the end of the oscillation cycle that forces one of these variables to fulfill the constraint. The constraint nodes were picked from among the nodes with the lowest oscillation frequencies. The globally optimal solution is thus stable as the variable(s) fulfilling a constraint will always be able to generate at lease one event that puts the constraint node in a fulfilled state during each cycle of the constraint node.

The above scheme is implemented by routing events according to Fig. 7.8a. Events from variable nodes can not dislodge a constraint node from the fulfilled state or state 4. Note that state 4 of a constraint node is the lowest priority state according to the state update function in Eq. 7.3 so an input event to a constraint node which encodes that state 4 and state k ($k \in \{1, 2, 3\}$) are allowed will always put the constraint node in state k, if it was not already at state 4. If a variable appears with the same sign in multiple constraints (negated or non-negated in all of them), an event generated by one of these constraint nodes that forces this common variable

to go to a fulfilling state will automatically fulfill the other constraints as well so I route such events to the other constraints so as to move them to the fulfilled state as shown in Fig. 7.8 and prevent them from unnecessarily flipping other variables. This scheme for solving 3-SAT is less powerful than the previously described approach based on the probSAT algorithm. At the end of the cycle of an unfulfilled constraint node, the constraint node simply flips the last variable in its domain to generate an event. Due to the continuously shifting phase relations, the choice of which variable to flip is essentially done "at random" with no regard for how many other constraints would be violated due to this flip. Figure 7.8b shows a histogram of the average number of oscillation cycles per variable needed to find the solution of an example 3-SAT problem.



(a) (b)

Figure 7.8: (a) Network showing the implementation of $C1 \wedge C2$ where $C1 = (L1 \vee L2 \vee \neg L3)$ and $C2 = (L2 \vee L3 \vee L4)$. Numbered squares indicate the output ports and, indirectly, the input ports of a variable and arrows indicate how events are routed. For example, events from port 1 of $L1$ go to input port 9('1001' in binary) of $C1$ which instructs $C1$ to go to state 4 or state 1. Events from port 2 of $L1$ go to port 8 of $C1$ which instructs $C1$ to go to state 4. (a) Histogram of the number of oscillation cycles (averaged per variable) needed by the chip to find the solution of a 3-SAT problem with 50 variables and 218 clauses over 1000 trials taken from (Hoos & Stützle, 1998)

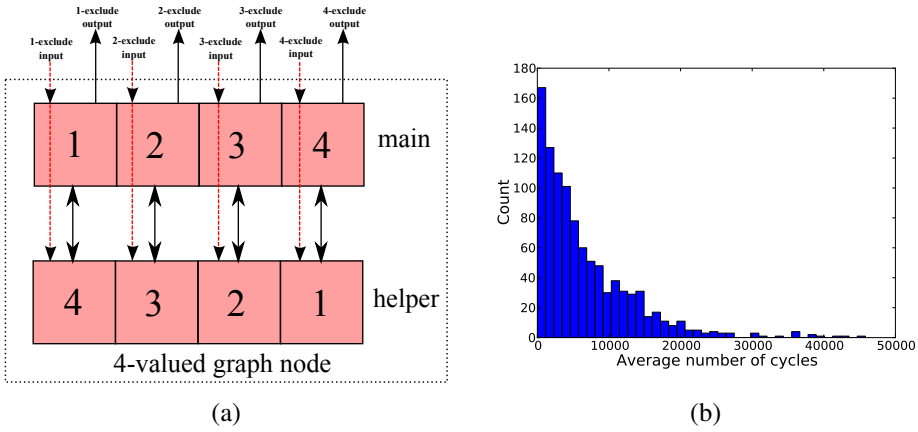
7.5.2. Solving Graph Coloring Problems on the Hardware Prototype

The hardware prototype can solve graph coloring problems with up to 8 colors. We can use a simple scheme where a graph vertex is represented by a chip node and events from port p of a node (which indicates that the node is in state/color p) go to input port $2^n - 1 - 2^{p-1}$ (all 1s binary string except at position p; p index starts from 1) of all adjacent nodes/vertices in the graph. I call these input ports the i-exclude input ports as receiving an event on them instructs the node to go to any state except p, thereby enforcing the constraint. However, this scheme will not work since a node always goes to an allowed state that has the lowest index when responding to an exclude event (see Eq. 7.3). All the nodes would thus quickly get stuck in the 1 and 2 states as the 1-exclude and 2-exclude events that the nodes send to each other will not be able to move any node out of these 2 states. I use the more elaborate scheme shown in Fig. 7.9a where two 4-valued chip nodes are used to implement one 4-valued graph vertex. The value of this graph vertex is index of the last event emitted by the 'main' chip node.

Pair-wise inequality constraints are implemented by routing events from the i-exclude output port of one vertex to the i-exclude input port of the other vertex. Assume a vertex has value 1, i.e, the state of the main (helper) chip nodes are 1 (4). The state/color of this vertex will only change if it receives an event on the 1-exclude port. In that case, the 'main' and 'helper' chip nodes go to states 2 and 1 respectively since these are the lowest index allowed states in the two chip nodes. The two chip nodes now have inconsistent states and whichever of them generates an event first forces the other node to switch its state; for example if the 'helper' node generates an event first, it forces the 'main' node to take state 4. A 1-exclude input event effectively has a 50% chance of moving this graph node to state 2 and a 50% chance to move it to state 4 due to the irregular phase relations.

The scheme can be extended to 6- and 8-valued vertices by using three 6-valued and four 8-valued chip nodes respectively to represent a single graph vertex and it is straightforward to show that using this scheme, the network representing the coloring graph always uses all available colors. 3-, 5-, and 7-coloring problems can be implemented by adjusting the even color schemes so that events are routed to input ports that exclude both the color/index of the source output port as well as the highest index/color which will then be unused.

One difficult graph for this architecture is the '5×5 queen' graph whose solution is equivalent to finding the non-interfering positions of 5 queens on a 5×5 chess board. The average number of cycles needed to find a solution is shown in Fig. 7.9b.



(a) (b)

Figure 7.9: (a) Implementation of a 4-valued graph vertex using two 4-valued chip nodes which are coupled so that an event from port 1, 2, 3, or 4 of one chip node puts the other node in state 4, 3, 2, or 1 respectively. This vertex receives events from other vertices which go the exclude input ports of the two chip nodes (red dashed lines). For example an event arriving on the 1-exclude input port goes to port 14(binary '1110') on the 'main' chip node and port 7(binary '0111') on the 'helper' chip node. (a) Histogram of the number of oscillation cycles (averaged per variable) needed by the chip to find the optimal coloring of the 5×5 queen graph.

7.6. Discussion

CSPs have often been examined through the lens of statistical physics (Mézard et al., 2002; Krzakala & Kurchan, 2007). Within the framework of statistical physics, a CSP is formulated as a distributed system that seeks to minimize the number of frustrated interactions (violated constraints) between its elements. Direct analogies can be established between the ground en-

ergy states of physical systems (where frustrated interactions are at a minimum) and solutions to CSPs (Barahona, 1982). The *inferenceEngine* architecture is fundamentally different from the systems analyzed in the framework of statistical physics, yet it captures some of the general features of such systems: The architecture makes use of a large number of locally interacting elements that mutually constrain each other so that the system as a whole tries to go to states where the number of frustrated interactions is at a minimum.

Perhaps the most distinguishing feature of the *inferenceEngine* architecture is the mechanism used to explore the solution space. In lieu of random fluctuations, the continuously changing phase relations between incommensurable oscillators are a source of non-repeating fluctuations that can be easily exploited in our event based architecture to realize efficient search algorithms. Pseudo-random number generators (PRNGs) could have been used to drive the search. Even assuming we could implement an efficient PRNG with a per-node complexity equivalent to an analog oscillator, PRNGs would require a clock. In each clock cycle, the PRNG scheme could choose one variable/constraint to update at random. This sequential scheme will fail to exploit the distributed and highly local nature of many CSPs (A constraint involves only few variables and a variable is part of only few constraints). This distributed and local nature is precisely what I am trying to exploit with the independent nodes running in parallel and trying to attain consistency in their local neighborhoods in the constraint graph. The PRNG scheme could update multiple, randomly chosen, variables/constraints per clock cycle. Such batch updates would go against the idea of probSAT, and stochastic local search algorithms in general, which make local moves that change only one variable at a time and propagate its new value before updating the next variable. Since nodes in a distributed architecture need to communicate, by having them communicate in an event-based manner at times governed by simple local analog oscillators, we obtain (almost for free) non-repeating fluctuations that could be easily exploited to drive a stochastic-like search.

While the architecture is general enough to allow the instantiation of various algorithms for solving CSPs, it is best suited to implementing algorithms of the local search variety in which each variable is iteratively updated based only on local information, i.e, based on the state of the constraints in which it is involved.

The digital event-based nature of node communication is key to the architecture's scalability and configurability. These digital pulses can be transmitted and routed using a digital fabric that links together a large number of nodes. In the prototype chip, event routing is done off-chip in a serial manner on an FPGA board (Fasnacht & Indiveri, 2011). This introduces a serial bottleneck in the otherwise massively parallel operation of the architecture. This serial bottleneck must be eliminated to reap the advantages of the massively parallel operation of the distributed architecture. A distributed event routing architecture that exploits the local nature of event communication (which reflects the local nature of the constraint graphs of many relevant problems) to route events in parallel in multiple local domain is necessary. Configurable and parallel AER routing fabrics have been proposed for use in large-scale neuromorphic systems (Joshi et al., 2010; Merolla et al., 2014a) and could be directly adapted for use in an implementation of the described architecture. As shown when solving SAT problems, the architecture is robust to event delays and lost events which relaxes the requirements on the event routing fabric.

In simulation we showed for the case of random SAT problems that the *inferenceEngine* architecture can run at a surprisingly slow mean oscillation frequency (around 2 MHz) and still attain a time to solution that is comparable to a CPU running at three orders of magnitude higher clock rate. The simple logic operations in the constraint and literal nodes can certainly run at such slow frequencies. These results indicate that the *inferenceEngine* architecture is a more efficient approach to solving SAT problems than conventional CPUs.

Algorithms for solving CSPs are often conceived with the digital von Neumann model of

computation in mind. The results presented in this chapter highlight an alternative approach which starts with no prior assumptions about the computational model, and seeks to exploit the physical characteristics of the underlying substrate in order to find a solution tailored to the computational problem at hand. In this chapter for example, I exploited the natural incommensurability of physical analog oscillators to derive a distributed novel algorithm for solving CSPs. The physical algorithms developed using this method naturally admit an efficient implementation on the physical substrate that underlied their derivation. The computing architectures developed using this bottom-up approach, such as the VLSI device I present in this chapter, have the potential to achieve considerable performance gains in their target problems compared to conventional purely digital approaches (Indiveri & Liu, 2015).

In the next chapter, I describe how the quasi-periodic behavior of the *inferenceEngine* architecture can be used to approximate the dynamics of stochastic networks.

8

Stochastic Interpretation of Quasi-periodic Event-based Systems

Stochastic spiking neurons are often used in biological network models to account for the variability of neural responses (Holt et al., 1996). Stochastic neural responses can have important computational implications such as allowing neural networks to sample from probability distributions (Buesing et al., 2011) or to transmit small subthreshold input signals (Longtin, 1993). In machine learning, networks using neuron-like units with stochastic activation functions are often used to realize probabilistic generative models of input data (Smolensky, 1986; Salakhutdinov & Hinton, 2009). Many multi-layer networks used in classification and encoding tasks (Hinton & Salakhutdinov, 2006; Vincent et al., 2010; Hinton et al., 2006) also make use of stochastic neuron elements to limit the amount of information that flows from one layer to another during unsupervised pre-training. This is important for allowing the neuron elements to learn useful features.

An open question is how such stochastic networks can be realized efficiently on custom silicon chips. Custom chip implementations of either machine learning network architectures (Pham et al., 2012) or the more biologically inspired spiking networks (Merolla et al., 2014b) can offer significant performance gains compared to simulating these networks on conventional general-purpose CPUs or GPUs. Straightforward implementations of stochastic networks would use an explicit true- or pseudo-random noise source in each unit to realize uncorrelated fluctuations. This would be inefficient as the number of units scales up since these "noise circuits" are not cheap in terms of power and silicon area.

This chapter describes how the *inferenceEngine* architecture described in the previous chapter can be used as an efficient, distributed, and easily implementable scheme for the generation of largely uncorrelated fluctuations in a large number of neuron elements. The scheme exploits the non-repeating phase relations in the *inferenceEngine* architecture. Each neuron element has access to the state of an analog oscillator. Due to the inevitable inhomogeneities in the silicon fabrication process, the different oscillators are guaranteed to be incommensurable, i.e, have oscillation frequencies that are not rational multiples of each other. The phase relations between these oscillators varies irregularly in an aperiodic manner. By developing the communication scheme between the neuron elements so that the interaction strength between a group of neurons depends both on the weights between them as well as the phase relations within the group, we can obtain neural activity that changes in a non-repeating manner and that can be modelled in stochastic terms.

In this chapter, I present a stochastic formulation that matches the 'statistics' of this quasi-periodic system and that replaces each deterministic neuron with an equivalent neuron having a stochastic activation function. I investigate the fidelity of this approximation and evaluate how the quasi-periodic system performance differs from a system that uses high quality pseudo-random number generators.

8.1. Sigmoidal Units

Figure 8.1a shows the general structure of a neuron. A neuron corresponds to a node in the *inferenceEngine* architecture described in the previous chapter. It is composed of a FSM in which events arriving on the inputs ports $a, b, c, \ldots$ trigger state transitions. Associated with the FSM is an analog oscillator that generates a regular train of events. The neuron routes each event from this internal oscillator to one of the output ports. If the FSM is in state Si/j when the analog oscillator generates an event, this event is routed to output port j. A neuron with k output ports can transmit $log_2 k$ bits of information in each oscillator cycle. We consider the index of the output port on which a neuron last generated an event as the neuron's current value. Neurons can be connected together (output ports to input ports) and we assume the oscillator frequencies in the different neurons are incommensurable.

Consider the FSM of a simple neuron, neuron T, that is shown in the top part of Fig. 8.1b. This neuron receives a stream of 1 events from two neurons and a stream of 0 events from one neuron. From the FSM of T, it is clear that T will route the internal oscillator events to the 1 (0) port if the last event it received was 1 (0) as shown in Fig. 8.1b. Even though the values of the three source neurons are constant, eventually, the 0 input event arrives just before the end of the cycle of T, and the value of T changes from 1 to 0. The value of a neuron (the identity of its last output event) thus does not solely depend on the values of the source neurons but it also crucially depends on the order of arrival of their events. This order is continuously changing in an aperiodic manner due the incommensurable oscillation frequencies of the oscillators inside the neurons.

Figure 8.1c shows the FSM of a slightly more complex neuron that also has two input ports and two output ports. I refer to neurons having such a counter-like FSM as sigmoidal neurons. Assume this neuron is receiving events from N constant-value source neurons as shown in Fig. 8.1d where x of the source neurons are sending 1 events and $N - x$ are sending 0 events. Figure 8.1e shows that the fraction of 1 events generated by the sigmoidal neuron of Fig. 8.1c is a sigmoidal function of x/N or the fraction of 1 events impinging on the neuron (the neuron has to generate either a 1 or a 0 event for each event from its internal oscillator). The shape of the activation function in Fig. 8.1e is robust to changes in N. It, however, crucially depends on the depth of the counter in the sigmoidal neuron as shown in Fig. 8.1f. The FSM of the sigmoidal neuron shown in Fig. 8.1c has depth 3 while the FSM of the neuron in Fig. 8.1b has depth 1.

To accurately calculate the fraction of 1 events generated by the target sigmoidal neuron, we need to enumerate all possible orderings of the N source neurons' events within one cycle of the target neuron and calculate what fraction of these orderings will put the target neuron's FSM in one of the 1 output states (the yellow states). The calculation will depend on the target neuron's initial state and is further complicated by the difference in oscillator frequencies which might lead to some source neurons not generating any events, or generating multiple events, during one cycle of the target neuron. The analysis can be greatly simplified if we interpret the system behavior in stochastic terms. At the heart of this stochastic interpretation is the following approximation:

The periodic train of events generated by a neuron is treated as a Poisson train

This approximation is similar to the stochastic approximation previously used to analyze the behavior of quasi-periodic winner-take-all networks Mostafa et al. (2015c); Muller (2015). Under the Poisson event generation assumption, the probability at any time of the next input event being 1 is x/N (the fraction of 1 source neurons) and of being0 is $1 - x/N$. The FSM of a neuron can thus be cast as a Markov chain having the same structure but with each 1 edge replaced by an edge with transition probability x/N and each 0 edge by an edge with transition probability $1 - x/N$. The probability of a particular output is the sum of the probabilities of

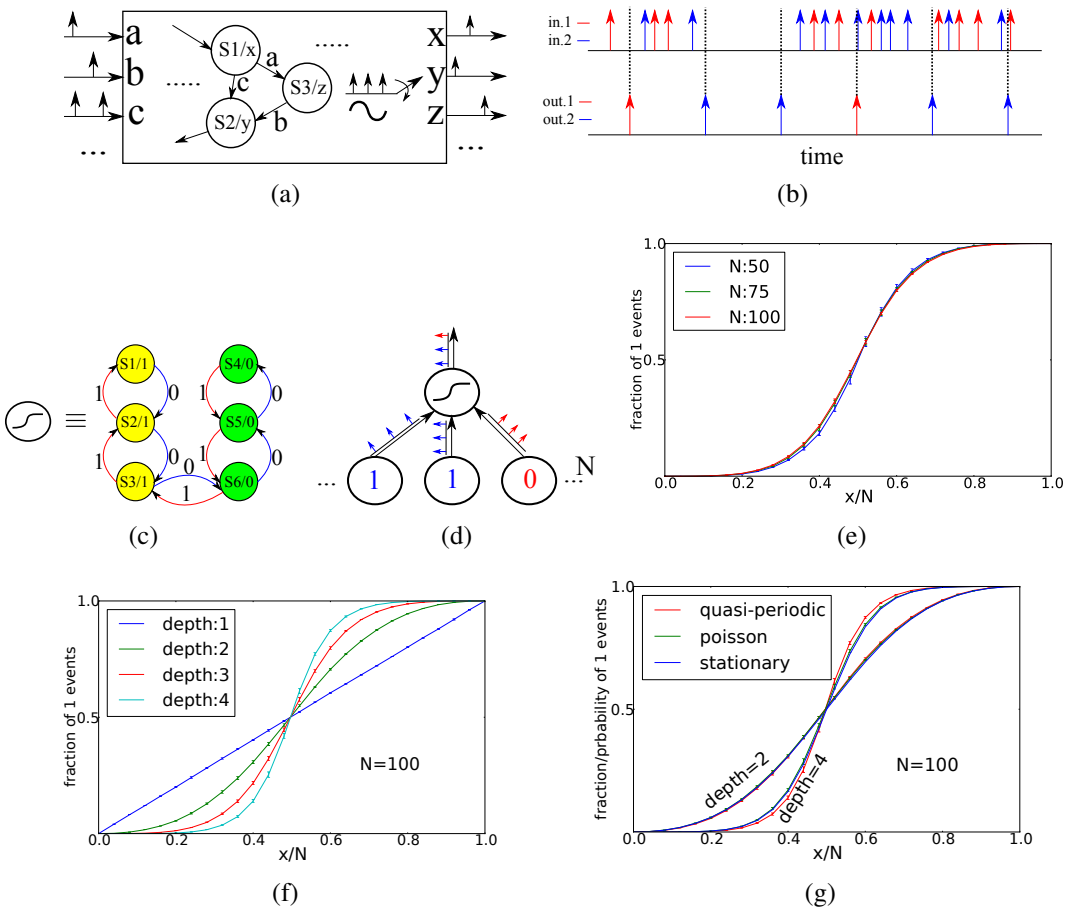
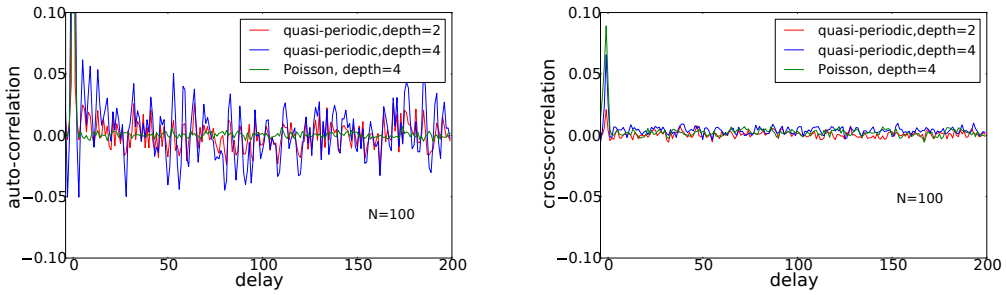


Figure 8.1: (a) General form of a neuron element. Arrival of input events trigger FSM transitions. When an event is generated by the internal oscillator, it is routed to one output port selected based on the current state of the neuron. (b) A simple two state neuron element and an illustrative simulation. (c) A neuron with a counter-like FSM. (d) The neuron from c receiving events from x neurons with value 1 and $N - x$ neurons with value 0. (e) Fraction of 1 events generated by the target neuron in d after $1.5 * 10^5$ events as a function of x/N (the fraction of afferent 1 neurons). (f) Shape of the target neuron activation function crucially depends on the depth of its counter/FSM. (g)Activation functions when each neuron generates a periodic train of events (quasi-periodic) or a Poisson train (Poisson), or calculated directly from the stationary distribution of the neuron-equivalent Markov chain (stationary). Lines in e, f, and g are averages over 5 trials. Error bars are the standard deviation. Oscillator frequencies redrawn in each trial uniformly from the range $[40, 50]$ Hz.

the states yielding that output in the Markov chain stationary distribution. For each x/N value, we can thus calculate the stationary distribution of the Markov chain and obtain the probability of generating a 1 event. This is plotted in Fig. 8.1g together with the activation functions of the deterministic quasi-periodic system and the stochastic system where the oscillators generate a Poisson (instead of a periodic) train of events. The stochastic approximation is valid but becomes less accurate as the depth of the sigmoidal neuron FSM/counter increases. That is because the state of a deeper counter reflects a longer history of events and in that longer history, differences between the periodic and the Poissonian event generation mechanisms become ap-



(a) (b)

Figure 8.2: (a) Part of the auto-correlation of the sequence of $1.5 * 10^5$ bits generated by a target neuron when $x/N = 0.5$ in the periodic and Poissonian event generation cases . (b) Cross-correlation of 2 sequences of bits generated by two target neurons receiving the same input when $x/N = 0.5$. Cross-correlation was calculated after adjusting the sequence of bits generated by one neuron so that the temporally closest events/bits in the two sequences occur in the same sequence position.

parent. For instance, the Poissonian neuron can generates two events in quick succession which is impossible in the periodic case.

The sequence of bits/events generated by the target neuron in the quasi-periodic system in Fig. 8.1d has small but non-decaying correlations as shown in Fig. 8.2a. A deeper counter (longer memory) in the target neuron results in higher correlations as it becomes more difficult for incoming events to yield a state/value at the cycle's end that is independent of the state at the cycle's beginning (which reflects the previous bit value). The non-decaying and non-repeating correlation structure is a product of the quasi-periodic nature of the system which causes the phase relations among the oscillators to almost repeat after a while and yield similar event orderings. Two target neurons having different frequencies and receiving events from the same neurons generate sequences of bits with small cross-correlation which is on par with the Poissonian system as shown in Fig. 8.2b.

In summary, the quasi-periodic event-based dynamics of the *inferenceEngine* architecture admit a stochastic interpretation in which event generation in each neuron/node is assumed to be Poissonian instead of periodic. This interpretation works because of the finite memory in each neuron which renders its output sensitive to the order of arrival of input events. Since this order changes in an irregular manner, the target neuron can see it as "random". Empirically, I observe that the stochastic approximation is accurate for other types of neurons/FSMs as long as the number of source neurons is large compared to the number of states in the target neuron's FSM.

8.2. Restricted Boltzmann Machines

The behavior of a network of neurons, where each neuron has the form shown in Fig. 8.1c can be interpreted in probabilistic sampling terms. The network connectivity (the way events are routed) induces a 'probability distribution' over the possible states of the network. In a Gibbs sampling fashion, we interpret each event from a neuron as a sample drawn from the distribution over this neuron's values conditioned on the current state of all other neurons. Under this interpretation, I show that the quasi-periodic network can reproduce the sampling behavior of a stochastic RBM.

An RBM is a Markov random field on a bipartite graph of binary 0/1 units. The graph has N_v visible units and N_h hidden units. Visible and hidden units are bidirectionally connected according to the $N_v * N_h$ weight matrix W. There are no connections between visible units or between hidden units. $\mathbf{x}$ and $\mathbf{y}$ are the visible and hidden bias vectors respectively. $\mathbf{v}$ and $\mathbf{h}$ are the vectors representing the states of the visible and hidden units respectively. The probability of a particular configuration is:

$$P(\mathbf{v},\mathbf{h}) = \frac{e^{\frac{-E(\mathbf{v},\mathbf{h})}{T}}}{Z} \tag{8.1a}$$

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{x}^T\mathbf{v} - \mathbf{y}^T\mathbf{h} - \mathbf{v}^T W\mathbf{h} \qquad Z = \sum_{\mathbf{v},\mathbf{h}} e^{\frac{-E(\mathbf{v},\mathbf{h})}{T}} \tag{8.1b}$$

where $E(\mathbf{v},\mathbf{h})$ is the energy of configuration $[\mathbf{v},\mathbf{h}]$, T the model temperature, and Z the normalizing constant or partition function. Generating samples from the distribution in Eq. 8.1 is typically done through Gibbs sampling which updates a visible(hidden) unit conditioned on the current state of the hidden(visible) units by drawing a sample from:

$$P(v_i = 1|\mathbf{h}) = \sigma_T(\sum_{j=1}^{N_h} w_{ij}h_j + x_i) \qquad P(h_j = 1|\mathbf{v}) = \sigma_T(\sum_{i=1}^{N_v} w_{ij}v_i + y_j) \tag{8.2}$$

where $\sigma_T(x) = 1/(1+e^{-x/T})$ is the logistic sigmoid function. A hidden or visible unit/neuron thus needs to have a logistic sigmoid stochastic activation function so that its output corresponds to a Gibbs sampling update step.

Figure 8.3a shows that the activation function of a sigmoidal neuron with depth 3 closely matches that of a logistic sigmoid. As shown in Fig. 8.1e, the shape of the activation function is robust to the number of source neurons. The sigmoidal neuron shown in Fig. 8.1c could thus be used to represent a unit in an RBM. The sigmoidal neuron's activation function is plotted as a function of the excess fraction of incoming 1 events: $x/N - 0.5$. For each neuron, we thus need to make the quantity $x/N - 0.5$ equal to a weighted sum of the states of the source units, plus a bias term as in Eq. 8.2.

Due to the discrete nature of the neurons, I can only use discrete weights and biases. To implement an RBM using 5 possible weights/biases: $\{-2,-1,0,1,2\}$, I need to use the neuron shown in Fig. 8.3b to implement each RBM unit. The neuron is similar to the one in Fig. 8.1c except that it has 4 independent oscillators that each generates a periodic train of events. The events from each oscillator can go to one of two output ports based on the state of the neuron's FSM. Thus, the neuron generates four event streams. If the neuron's state is $S1$, $S2$, or $S3$. The events from the 4 oscillators are routed to output ports $a1$, $b1$, $c1$, and $d1$, otherwise they are routed to $a0$, $b0$, $c0$, and $d0$. Figure 8.3c is an example of how to connect neurons/units to implement weighted connections with discrete weights. Only the visible to hidden connections (which are a mirror of the hidden to visible connections) and the hidden biases are shown. The four 0 event streams from a visible unit are distributed equally on the 0 and 1 input ports of the target neurons. The four 1 event streams of a visible unit are distributed on each target neuron's input ports so as to implement weighted connections. The connection scheme ensures that regardless of the state of the visible units, the number of incoming event streams at a hidden units (N) is always 12. The hidden biases are implemented as weighted connections from an always 1 neuron.

It is easy to verify in Fig. 8.3c that the excess fraction of incoming 1 events is $(2v0 - v1 - 1)/12$ and $(-2v0 + 2)/12$ for hidden units/neurons $h0$ and $h1$ respectively (the weights are scaled by a constant factor). Arbitrary RBMs with weights/biases in the range $\{-2,-1,0,1,2\}$

can be similarly implemented. I implemented an RBM with 10 visible and 10 hidden units whose weights and biases are drawn randomly from the integers between -3 and 3 (Each neuron/unit has 6 oscillators and 6 output event streams) . The RBM is small enough to enable the numerical calculation of the probabilities of each of the 2^{20} configurations. Figure 8.3c shows the evolution of the KL divergence between the sample distribution and the true RBM distribution in two cases: when the samples are generated from a conventional RBM using Gibbs sampling and when they are generated from the quasi-periodic network implementation. In the quasi-periodic network, sampling is done in a decentralized manner. Each neuron/unit generates an event/sample whenever one of its internal oscillators generates an event. The latest event/sample generated by each unit defines the current state of the network. A new sample is obtained as soon as each oscillator has generated at least one event since the last sample was recorded.

As the number of samples increases, the sampling distribution approaches the true RBM distribution in both cases as shown in Fig. 8.3d. The quasi-periodic network sampling distribution eventually becomes quite close to the true RBM distribution, yet not as close as the Gibbs sampler distribution. I believe this is because the neuron/unit activation function is not exactly a logistic sigmoid (Fig. 8.3a) which renders the quasi-periodic network distribution slightly different from the ideal distribution in Eq. 8.1. This shows, however, that a quasi-periodic event-based system can closely approximate a Gibbs sampler.

8.3. Stochastic Resonance

The simulations and data analysis in this section were carried out by Lorenz Muller.

Leaky integrate and fire (LIF) neurons receiving Poisson background noise (Plesser & Gerstner, 2000) have become a popular substrate for implementing stochastic computation in neural circuits (e.g. (Buesing et al., 2011)). We study the phenomenon of stochastic resonance (Benzi et al., 1981) in LIF neurons driven by periodic, incommensurable spike trains and compare this situation to Poisson background noise. The LIF neuron model we use (similar to the model of (Gerstein & Mandelbrot, 1964)) contains a single counter (the 'membrane potential'); when that counter reaches a threshold (ten) the neuron fires and resets the counter to zero. The leak is implemented by a periodic input of weight -1 and frequency 200 Hz.

We examine the output of such a LIF neuron x in the following stochastic resonance scenario (see figure 8.4a): x receives periodic input from a source neuron at a fixed rate of 30 Hz with weight 6, which is too low to elicit spikes from x. x additionally receives input from various background sources; this background input sometimes brings x close to the threshold and can enable x to spike in response to the input from the source neuron. We characterize the output of neuron x by the positive predictive value (ppv) of its spike train with respect to the spike train it receives from the source neuron, S. The ppv is the number of spikes from x produced within 1 ms of a spike from S, divided by the total number of spikes from x.

We study two different settings for the background input activity to neuron x. Total average rate of background spike activity is r_{tot} in both settings. In each setting, spikes from half the background sources have weight 1, and spikes from the other half have weight -1. In the first setting, two ($+$ and $-$) Poisson spike trains with rate r_{tot} each represent the background activity. In the second setting, background activity is represented by two ($+$ and $-$) collections of spiking oscillators with n oscillators in each collection. Each oscillator has rate z/n where z is a uniform random variable drawn from the range $[0.95 \cdot r_{tot}, 1.05 \cdot r_{tot}]$.

The stochastic resonance effect is shown in Fig. 8.4b. For Poisson input as well as for sufficiently large number of incommensurable periodic inputs, we observe the stochastic resonance

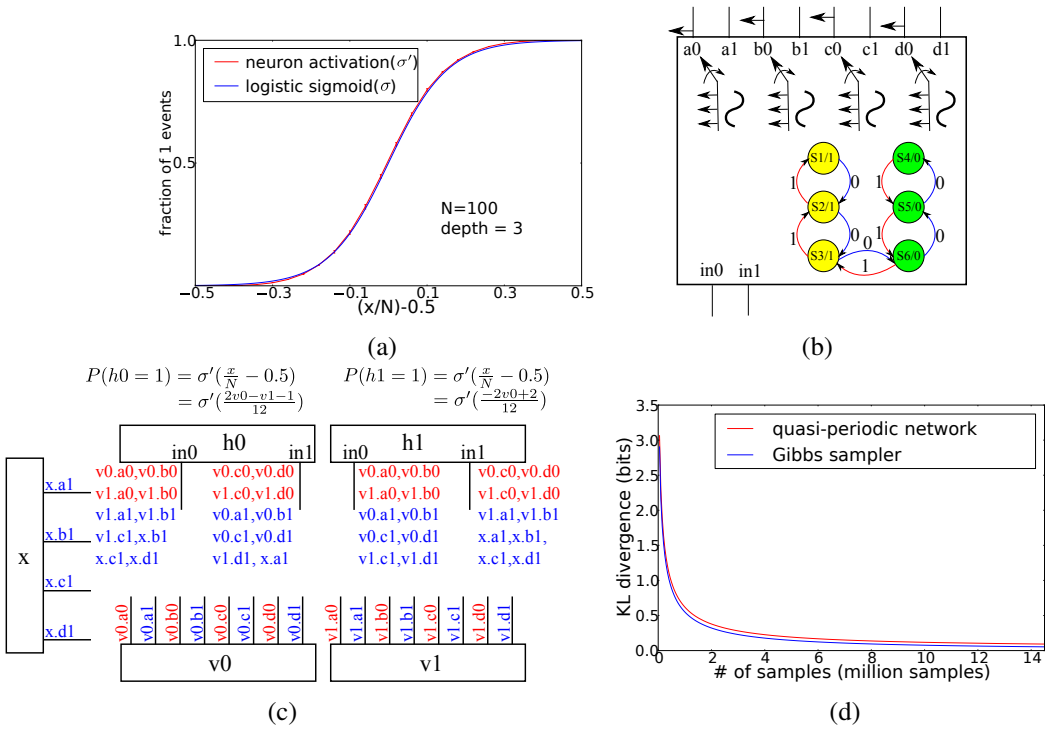
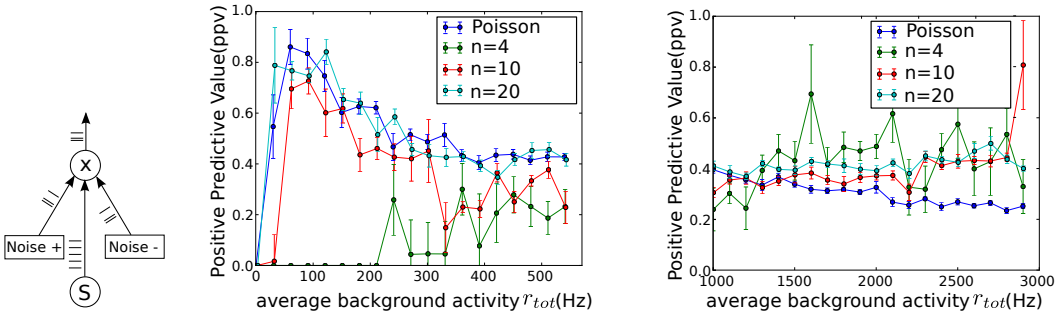


Figure 8.3: (a) The depth 3 neuron activation as a function of the excess fraction of input 1 events closely matches a logistic sigmoid function. (b) Implementation of a sigmoidal neuron with 4 output event streams. (c) Example RBM with weights $w_{00} = 2$, $w_{01} = -2$, $w_{10} = -1$, and $w_{11} = 0$, and hidden biases $x_0 = -1$ and $x_1 = 2$. Only visible to hidden connections and hidden biases are shown. Next to each input port of a hidden unit is a list of the output ports of the visible units whose events are routed to that input port. 0 output ports are in red, and 1 output ports are in blue. The probability of generating a 1 event for each hidden neuron is shown. σ' is the sigmoidal neuron activation function shown in a. (d) KL divergence between the sampling distribution and the true probability distribution of an RBM when the samples are generated by a Gibbs sampler (blue) and by the quasi-periodic network implementation of the RBM (red). Simulation was repeated four times with different random RBMs and yielded virtually identical curves.

phenomenon. At high event rates for the oscillatory inputs, the ppv degrades less than in the Poisson case (Fig. 8.4c) because the oscillators produce 'balanced' input sequences; in a given time window, the Poisson inputs can push the counter of x arbitrarily far from its initial state for a sufficiently high r_{tot}. In the oscillatory case the counter of x can maximally stray n steps from its initial state if we assume the total spike rate of each collection ($+$ and $-$) are equal, irrespective of r_{tot}. This induces a narrower distribution on the membrane potential/counter value compared to the Poisson case.

8.4. Hardware Demonstration

I used the prototype chip described in the previous chapter (section 7.5) and which implements a simple version of the *inferenceEngine* architecture to realize simple sampling dynamics. Fig. 8.5a shows the structure of the network implemented on the chip. The network has 10 pat-



(a) (b) (c)

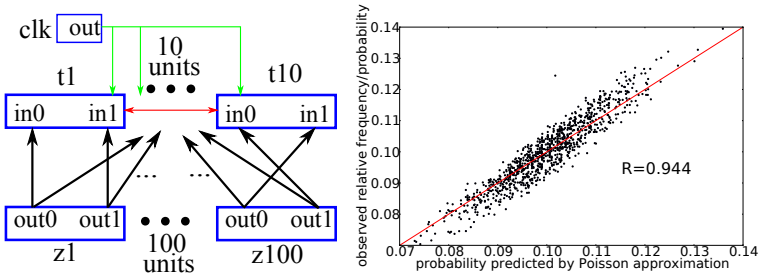
Figure 8.4: (a) Stochastic resonance network layout (b) Poisson input noise enables the neuron to respond to spikes from neuron S; when the Poisson rate is too large, ppv drops. A small number ($n = 4$) of random frequency oscillatory inputs do not yield the same effect. As the number of oscillatory inputs gets larger, the stochastic resonance phenomenon reemerges. (c) Oscillatory inputs cause ppv to degrade less at high background spike rates compared to the Poisson case. This is due to bounded excursions away from initial conditions in the oscillatory case. (see text).

tern units, $t1$ to $t10$. Each receives input events from 100 input units. The uni-directional connection from an input unit to a pattern unit can either have weight 1 (example $z1$ to $t1$) or weight 0 (example $z100$ to $t10$). Pattern unit i is thus associated with a binary weight vector $\mathbf{w}_i$ that defines its preferred pattern, i.e, the input unit values that will maximize its 'probability' of generating a 1 event.

Whenever a pattern unit generates a 1 event, it shuts down all pattern units (including itself). An event from the 'clk' unit activates the pattern units and sets them at state 0. Let the binary vector $\mathbf{z}$ denote the state of the input units. Define m_i as the number of matching entries in the vectors $\mathbf{w}_i$ and $\mathbf{z}$, divided by the vector lengths (100). Assuming the oscillator frequencies in the different units are not very different, the number of events arriving at the $in1$ port of pattern unit i in one oscillation cycle divided by the total number of received event, x_i/N, is on average equal to m_i. Assuming event generation in each unit is Poissonian instead of periodic, the probability of a pattern unit generating a one event $P(t_i = 1)$ is proportional to $x_i/N = m_i$ (see the linear activation function in Fig. 8.1f). But since pattern units are in a competitive configuration, only one pattern unit can generate a 1 event for each 'clk' event, the renormalized $P(t_i = 1)$ is:

$$P(t_i = 1) = m_i / \sum_{j=1}^{10} m_j \tag{8.3}$$

In the chip experiment, each weight vector $\mathbf{w}_i$ was randomly initialized. 100 different input layer patterns $\mathbf{z_1} - \mathbf{z_{100}}$ were then applied to the input layer. The events of the pattern units were collected and $P(t_i)$ evaluated from the events/samples for each input pattern. The resulting 1000 data points are plotted in Fig. 8.5b as a function of the $P(t_i)$ values predicted by the Poisson assumption (Eq. 8.3). The discrepancy is largely because the Poisson approximation assumes all units generate a Poisson train with the same rate, while the frequencies of the physical oscillators are different, thus biasing the competition in favor of pattern units with higher frequencies. If the FSM in the pattern units were a depth 3 counter (as in Fig. 8.1c), then the competition would be between a number of units whose activation functions closely approximate the logistic sigmoid



(a) (b)

Figure 8.5: (a) Structure of the network implemented on the VLSI chip. A 1 event from a pattern unit ($t1$ to $t10$) shuts down all pattern units. A normal binary unit is designated as a clock unit ('clk') and its events reactivate the pattern units and puts them at state 0. (b) The observed relative frequencies of the 1 events from each pattern unit in the chip for each input pattern compared to the predictions of Eq. 8.3.

and $P(t_i = 1)$ would then approximately be the softmax function: $e^{m_i} / \sum_{j=1}^{10} e^{m_j}$.

8.5. Discussion

Many stochastic algorithms used in machine learning and optimization applications or as models of biological computation are formulated as a distributed stochastic network where each element integrates incoming messages/spikes, applies a stochastic non-linear transformation, then emits a message/spike. I have shown that these stochastic networks can be reformulated in a radically different way as a quasi-periodic event-based system. The combined effect of several periodic ,but incommensurable, event/message streams on a target neuron with limited memory can be formulated in stochastic terms by assuming the event streams are Poissonian instead of periodic. This allows the FSM in a neuron to be treated as a Markov chain that is then used to accurately approximate the relative frequencies of the occupancies of the different FSM states in the quasi-periodic system

The scheme I describe for realizing approximations of stochastic units is quite suitable for large distributed systems as noise-generating resources in each unit are not required. By simply changing the communication scheme so that messages/events are communicated in a decentralized quasi-periodic manner, good approximation of stochastic behavior can be obtained (see the comparison to Gibbs sampling in Fig. 8.3d). The stochastic resonance example demonstrates that stochastic spiking network models could also be cast within this quasi-periodic framework. One advantage of the proposed scheme is the ease by which different approximations of stochastic activation functions can be "programmed", simply by changing the form of the neuron's FSM (see Fig. 8.1f). By reformulating quasi-periodic event-based dynamics in stochastic terms, the results presented in this chapter highlight a new direction for physical implementations of distributed stochastic systems. They also highlight the versatility of the *inferenceEngine* architecture which was used in the previous chapter to solve CSPs and in this chapter to approximate the behavior of stochastic networks.

9

Discussion

Efforts for developing brain-inspired computing architectures face the difficult challenge of matching the performance of digital processors. Several decades of optimizations at the level of fabrication processes, circuit designs, and algorithms have rendered digital processors highly efficient at solving general computational problems. It is thus only realistic that investigations into novel computing architectures should focus on problem-specific architectures in order to attain state-of-the art efficiency on these problems. This is the approach that was followed in this thesis. My goal was not to develop a general computing architectures, but rather to develop an architecture that solves a particular class of problems at which conventional processors struggle: best-match problems (Rumelhart & McClelland, 1986; Rumelhart, 1998) in which the goal is to find the values of a large number of variables so as to maximally satisfy a set of constraints.

In order to develop a physical architecture for solving best-match problems, I took biological networks as a source of inspiration. This was motivated by the distributed and massively parallel operation of biological networks which naturally reflects the distributed nature of best-match problems. The fact that neural networks are a natural computing substrate for solving best-match problems was early recognized (Rumelhart & McClelland, 1986), yet until now, no computing architecture was developed based on this insight so as to provide a viable alternative to conventional processors in solving best-match problems.

The *inferenceEngine* architecture that I developed to solve best-match problems was inspired by the features of Gamma-band rhythmic inhibition in the brain and how this rhythmic inhibition dynamically modulates the interaction strengths between neural groups. In chapter 4, I investigated the functional implications of Gamma-band rhythmic inhibition and how it allows neural networks to search for maximally consistent states, thereby providing a mechanistic framework for modelling various perceptual inference phenomenon. The *inferenceEngine* architecture described in chapter 7, however, looks quite different from the networks presented in chapter 4. Instead of using WTAs, the *inferenceEngine* architecture uses FSMs with configurable transition functions; instead of analog communication between the variables/WTAs, the nodes in the *inferenceEngine* architecture communicate in an event-based manner at times governed by local oscillators. The *inferenceEngine* architecture only retains the central dynamical feature of the biologically-inspired networks: the modulation of effective connectivity in a non-repeating fashion through the action of incommensurable oscillators.

Only by radically reformulating the dynamics of the biologically-motivated networks in chapter 4 was it possible to obtain an efficient VLSI realization. This fact illustrates the importance of taking biologically-motivated models only as a source of inspiration and not as models that should be faithfully emulated on silicon.

9.1. Efficient Computation and Substrate Dependence

The intricacies of the computational substrate do not factor into the formal treatment of computation (Sipser, 1996). Formal treatment of algorithms and their space and time complexity are carried out in relation to idealized computing models like Turing machines. Even though these formal treatments refer to the abstract Turing machine model of computation, they can provide general guidelines and constraints regarding the computational efficiency of any physical system, since it is believed that physical laws are effectively computable (Deutsch, 1985). This makes it highly unlikely that the time and space requirements of a class of problems (in relation to a Turing machine) could be circumvented by a cleverly designed physical system. For example, similar arguments have been used to argue that no physical system can solve NP-complete problems in polynomial time (Aaronson, 2005) (since a Turing machine presumably can not).

One of the main assumptions and motivations of this thesis was that alternative physical computing substrates could solve a particular class of problems (best-match problems) more efficiently than conventional computers. Of course, this does not mean that alternate physical schemes can solve problems in this class (which are typically NP-hard) in polynomial time, but rather that a physical system could naturally exhibit search dynamics that can effectively find solutions to best-match problems more quickly than a conventional processor executing a search algorithm. This assumption was justified through the *inferenceEngine* architecture.

It has been argued that the computations carried out by the brain are substrate-independent (Bostrom, 2003). However, it is precisely the substrate-dependence of these computations that make them interesting to neuromorphic engineers. It is the co-evolution of "hardware" (the physical neurons and synapses and the thousands of signaling pathways they employ) and "software" (neural and synaptic dynamics and their behavioral correlates) under strict survival constraints in a changing environment that shaped animal brains into efficient computing devices. Therefore, I believe one should follow two guidelines when seeking to develop brain-inspired computing architecture:

1. One should not start with an a-priori defined mode of computation or algorithm from the field of computer science or machine learning, and seek to engineer a neural network that implements this mode of computation. That is because these algorithms and models of computation were developed mainly with an eye to efficient implementation on digital computers. There is no reason to expect that a different computing substrate that is brain-inspired would match the performance of a digital processor when implementing these algorithms (which were primarily developed to run efficiently on digital computers). Consider the concrete example of developing neural networks to solve best-match problems. It is tempting to try to directly use algorithms that have proved their success in the machine learning literature such as belief propagation and MCMC sampling, among others, to explain how biological networks solve best-match problems. This, however, restricts the very wide space of possibilities in which a complex dynamical system like a biological network can handle best-match problems. Machine learning algorithms are usually analytically tractable and run efficiently on digital computers but this in no way recommends them as good models for how biological networks compute. In this thesis, I have eschewed prior assumptions about how a neural network can represent and manipulate probability distributions. Instead, I have shown that oscillatory inhibition that gives rise to rhythmic modulation of firing rate and rhythmic modulation of sensitivity to external inputs in the local circuitry allows a neural network to solve best-match problems in the general sense by searching for states that maximize the number of satisfied consistency conditions while remaining faithful to external input. The network dynamics can

not be exactly mapped to known inference algorithms in the machine learning literature. Yet the representation of probability distributions in these networks has many features in common with MCMC sampling. The dynamics of these networks can not also be exactly mapped to any known algorithms for solving constraint satisfaction problems (CSPs), yet these dynamics have a very close relation to stochastic local search algorithms, and empirically they can efficiently solve many kinds of CSPs.

2. When trying to map the networks developed according to the first guideline onto custom VLSI chips, one should again consider the issue of substrate-dependence. It is of course obvious that the standard VLSI fabrication process and the devices it provides to designers does not target the efficient implementation of neural networks. This makes it imperative that one should rethink the dynamics of the neural network to be implemented in order to match the strengths and limitations of the VLSI substrate. Otherwise, one again risks falling into the pitfall of implementing computing models and algorithms that were developed for one physical substrate onto a different substrate which would most likely lead to sub-optimal performance. I followed this guideline when developing the *inferenceEngine* architecture which adapts the dynamics of the biologically-motivated oscillatory networks so that they are compatible with a VLSI substrate

In summary, tight coupling between the algorithm/dynamics and the physical substrate is crucial for developing efficient computing systems. One reason neuromorphic architectures still lag behind in terms of performance on real-world problems might be because they are still designed or configured so as to implement algorithms developed for digital computers. To address this problem, one should take inspiration from the brain where structure and function seem to be closely interwined, and consider co-developing algorithms and architectures that take the strengths and limitations of the physical substrate into account.

9.2. Physical Interpretation of Best-match Problems

While all computational problems are eventually solved by a physical system, some problems map more naturally to the dynamics of simple systems than others. For example, in the thermodynamical limit, the exponential relation between the energy of a state and the probability of its occupancy follows directly from fundamental physical laws such as conservation of energy and entropy maximization (Kittel & Kroemer, 1980). Such fundamental physical behavior intrinsically solves an optimization problem where the system automatically gravitates to states that minimize the number of frustrated interactions between its components (Barahona, 1982). It is no surprise that such basic physical behavior provided the inspiration for optimization algorithms such as simulated annealing (Kirkpatrick et al., 1983) and survey propagation (Braunstein et al., 2005). This basic physical behavior and the algorithms it inspires are essentially solving best-match problems where the goal is to find a configuration that minimizes the number of violated constraints or frustrated interactions. The observation that simple physical systems naturally solve best-match problems is not merely of theoretical interest; Substantial investments have gone into developing quantum systems with programmable interactions (or constraints) so that the natural behavior of relaxation to ground states (states that minimize the number of violated interactions) effectively solves a combinatorial optimization problems (Johnson et al., 2011).

The fact that solving best-match problems has a direct analogy to the relaxation of a physical system to a minimum energy state (or set of states) makes these problems excellent candidates

when considering problems to be solved by non von-Neumann architectures. Since von Neumann architectures were not developed with an eye to solving this particular kind of problems, there is a good chance that an alternative physical system that exploits the distributed nature of these problems and their analogy to basic physical phenomena could outperform conventional processors when solving them. This was the motivation behind investigating this kind of problems in the context of biologically-motivated networks and the development of the *inferenceEngine* architecture.

9.3. Future Work

One of the promising aspects of neuromorphic systems is that they can process inputs in a highly parallel manner. However, having a dedicated circuit for each synapse and each neuron that enables them to operate in parallel is not enough to achieve parallel computation. To achieve that, a crucial requirement is that communication between neurons, i.e, the routing of spikes between the neurons should proceed in a parallel and distributed manner. The spike-routing or communication infrastructure in a neuromorphic system potentially plays a greater role in defining the speed and efficiency of the system compared to conventional digital multi-core architectures. That is because a neuromorphic system computes using a large number of simple processing elements/neurons. Due to their simplicity, these elements could run at very high speeds; this, and the fact that there are so many of them, places very tight requirements on the event-routing circuits to avoid having a communication bottleneck that throttles the system speed.

When testing the prototype chip implementing the *inferenceEngine* architecture, it was quite obvious that the serial routing of events was a fundamental bottleneck. This bottleneck forced me to reduce the oscillation frequencies of all nodes to avoid blocking the event-router, thereby reducing the speed at the which the chip solved CSPs. I thus believe that for an implementation of the *inferenceEngine* architecture to compete with state of the art processors when solving best-match problems, and for neuromorphic systems in general to match the speed and efficiency of digital approaches, an event-routing scheme is needed which is flexible (in the sense of being able to support a large number of network topologies) and massively parallel. Only then can we reap the speed and power advantages of the massively parallel operation of the processing elements/neurons. One of my future goals is thus to develop a distributed event-routing scheme that exploits any localities in the network topology to route events in parallel within a large number of local domains, thereby avoiding any central communication bottlenecks that could limit the speed of the network.

In chapter 4, I showed that oscillatory networks can be used to model perceptual inference phenomena. The network models that I presented generated testable predictions that could be used to experimentally assess whether neural oscillations do indeed play a part in perceptual inference phenomena in the brain. However, these model networks were quite abstract and used highly simplified representations of neural populations and of the Gamma-rhythm, which limited their predictive and explanatory power. In future work, I intend to develop more detailed and realistic models that use spiking neurons and that implement a bio-physically realistic model of Gamma-rhythm generation (such as the models in refs. (Jadi & Sejnowski, 2014; Brosch et al., 2002; Brunel & Wang, 2003; Gray & Singer, 1989)). The more realistic neural framework will allow me to model and investigate phenomena that were not accessible in the abstract model. I will be able to investigate the effect of input-induced Gamma-band coherence (Ray & Maunsell, 2010) on the network behavior and also the effect of the spontaneous formation of synchronized assemblies on how the network searches for consistent interpretations. I will

also investigate how input-induced coherence between different neural assemblies interacts with synaptic plasticity rules to modulate the strength of inter-assembly connections. I have shown that oscillatory networks can represent a probability distribution by sampling. I have also shown that the network can learn a simple consistency model by example. I intend to extend this into a general framework for learning the probability distribution of the input neural activity. After learning, the network should be able to produce samples from the learned distribution; in other words, the network should learn a probabilistic generative model of its input. Using the more detailed spiking model, I will be able to model more complex perceptual inference phenomena such as 'illusory contours' (der Heydt et al., 1984; Grosof et al., 1993).

Appendices

A

Analysis of Asymmetrically Coupled Rate-based WTA Circuits

In this appendix, I provide a simplified analysis of the network shown in Fig. 2.2. I will not investigate the evolution of the network's activity in time, but rather will derive constraints on the population parameters and the connection weights that enable the network shown in Fig. 2.2 to support stable sequential activity. The parameter set that I choose should ensure the elimination of the stable attractors in each individual WTA stage, and ensure that the following sequence of events takes place in the correct order in each WTA stage:

1. Activity in the WTA stage ramps up due to excitation from the previous stage until it becomes self-sustaining. A winner is selected.
2. Activity in the winning population reaches a level that enables it to shut down the previous stage through the inter-stage feedback excitatory-inhibitory connection.
3. Activity in the winning population reaches a level that enables it to activate the subsequent stage and push activity in the winning excitatory population in the subsequent stage beyond the self-sustaining rate.

Individual WTA Stage

Consider an individual WTA stage made up of a single excitatory population and a single inhibitory population and which is described by:

$$\tau_e \dot{x}^{exc}(t) + x^{exc}(t) = Max(0, w^{ee}x^{exc}(t) - w^{ie}x^{inh}(t) - T_e) \tag{A.1}$$

$$\tau_i \dot{x}^{inh}(t) + x^{inh}(t) = Max(0, w^{ei1}x^{exc}(t) - T_i) \tag{A.2}$$

We did not include multiple excitatory populations because we make the simplifying assumption that the winner selection happens so quickly in a WTA stage that all but the winning population become quickly suppressed and so only one excitatory population contributes to the dynamics of the WTA stage. We make the additional simplifying assumption that $\tau_i \ll \tau_e$ so that we can reduce the system (8-9) to the 1D system:

$$\tau_e \dot{x}^{exc}(t) + x^{exc}(t) = Max(0, w^{ee}x^{exc}(t) - w^{ie}Max(0, w^{ei1}x^{exc}(t) - T_i) - T_e) \tag{A.3}$$

The goal is to find the parameter set that will make this individual, uncoupled stage exhibit a non-zero stable attractor. This can be achieved if the phase space of the 1D system looks like Fig. A.1a. Each break in the plot corresponds to the second argument of one of the Max functions crossing zero. The necessary constraints are:

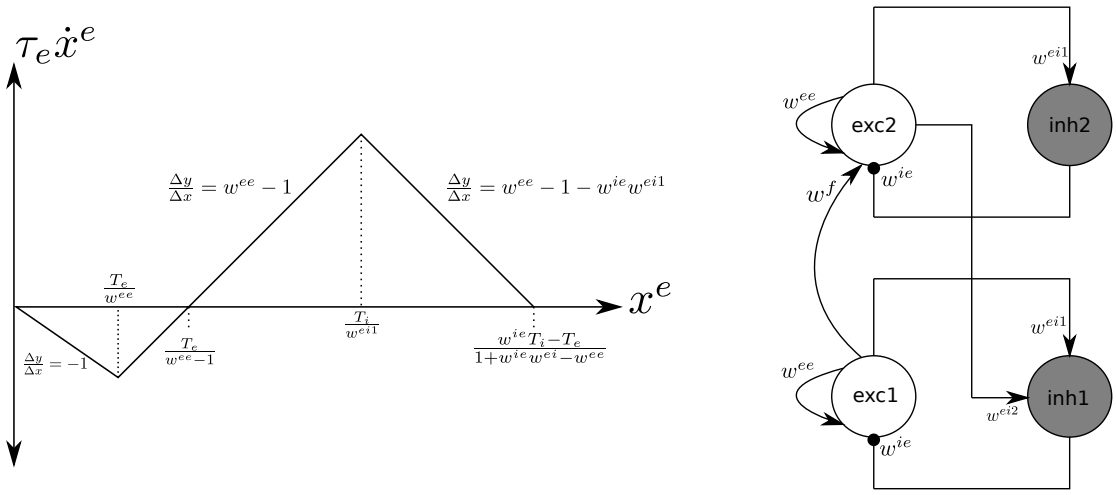


Figure A.1: (a) The 1D phase diagram of the system described by Eq. A.3. (b) Two asymmetrically coupled WTA stages. White circles are excitatory populations, gray circles are inhibitory populations

$$w^{ee} > 1 \tag{A.4}$$

$$w^{ee} - 1 - w^{ie} w^{ei1} < 0 \tag{A.5}$$

$$\frac{T_e}{w^{ee} - 1} < \frac{T_i}{w^{ei1}} \tag{A.6}$$

The self sustaining rate is $\frac{T_e}{w^{ee}-1}$. Beyond that rate, $\dot{x}^{exc}$ is positive and the excitatory population will approach the steady state activity in the absence of external input. The steady state activity is given by:

$$x_{SS} = \frac{w^{ie} T_i - T_e}{1 + w^{ie} w^{ei} - w^{ee}} \tag{A.7}$$

Coupled WTA Stages

Consider two WTA stages that are coupled as shown in Fig. A.1b.

$$\tau_e \dot{x}_1^{exc}(t) + x_1^{exc}(t) = Max(0, w^{ee} x_1^{exc}(t) - w^{ie} x_1^{inh}(t) - T_e) \tag{A.8}$$

$$\tau_i \dot{x}_1^{inh}(t) + x_1^{inh}(t) = Max(0, w^{ei1} x_1^{exc}(t) + w^{ei2} x_2^{exc}(t) - T_i) \tag{A.9}$$

$$\tau_e \dot{x}_2^{exc}(t) + x_2^{exc}(t) = Max(0, w^{ee} x_2^{exc}(t) + w^{f} x_1^{exc}(t) - w^{ie} x_2^{inh}(t) - T_e) \tag{A.10}$$

$$\tau_i \dot{x}_2^{inh}(t) + x_2^{inh}(t) = Max(0, w^{ei1} x_2^{exc}(t) - T_i) \tag{A.11}$$

Again, we make the simplifying assumption that $\tau_i \ll \tau_e$. Assume that we briefly excite the population x_1^{exc} so that it is approaching the attractor state of the uncoupled stage given by A.7. The next stage will be activated when the input to the population x_2^{exc} is above T_e, i.e., $w^f x_1^{exc} > T_e$. Hence, using Eq. A.7, we impose the constraint

$$\frac{T_e}{w^f} < \frac{w^{ie} T_i - T_e}{1 + w^{ie} w^{ei} - w^{ee}} \tag{A.12}$$

This ensures that the first(bottom) stage activates the second stage before the activity of the first stage settles into the attractor. Activation of the second stage will immediately increase the inhibition in the first stage. The magnitude of the non-zero steady state activity of x_1^{exc} as a function of x_2^{exc} assuming the latter is kept fixed can be written as

$$x_1^{exc}|_{SS} = \frac{w^{ie}(T_i - w^{ei2}x_2^{exc}) - T_e}{1 + w^{ie}w^{ei} - w^{ee}} \tag{A.13}$$

Using this result, we impose a stronger version of condition A.12 to ensure that the excitatory population in the first stage continues to provide superthreshold input to the second stage until the excitatory population in the second stage reaches the self-sustaining rate.

$$\frac{T_e}{w^f} < \frac{w^{ie}(T_i - w^{ei2}\frac{T_e}{w^{ee}-1}) - T_e}{1 + w^{ie}w^{ei} - w^{ee}} \tag{A.14}$$

In deriving the last condition, we have ignored the transients in x_1^{exc} and assumed that it immediately settles into the steady state given by Eq. A.13 in response to any change in x_2^{exc}. If we keep this assumption, then the increasing activity of x_2^{exc} as it ramps up beyond the self-sustaining rate will at some point drive the activity of x_1^{exc} below the self-sustaining rate, effectively extinguishing it. We denote the value of x_2^{exc} at which this happens by x^{SD}. Using (A.13), we can define x^{SD} implicitly:

$$\frac{w^{ie}(T_i - w^{ei2}x^{SD}) - T_e}{1 + w^{ie}w^{ei} - w^{ee}} = \frac{T_e}{w^{ee} - 1} \tag{A.15}$$

We impose the additional condition that a WTA stage should shut down the previous stage before activating the next one. Combining this additional condition with condition A.14 and using A.15

$$x^{SD} = \frac{T_i(w^{ee} - 1) - T_e w^{ei1}}{w^{ei2}(w^{ee} - 1)} < \frac{T_e}{w^f} < \frac{w^{ie}(T_i - w^{ei2}\frac{T_e}{w^{ee}-1}) - T_e}{1 + w^{ie}w^{ei} - w^{ee}} \tag{A.16}$$

We found that by satisfying the conditions A.4, A.5, A.6, and A.16 we could obtain well defined sequential activity. Complications may arise when there are multiple excitatory populations in a WTA stage that receive very similar inputs from the previous stage. In that case, our assumption that only one excitatory population contributes to the dynamics of a WTA stage fails. Multiple active excitatory populations in a single stage may cooperate to shut down the previous stage before any of them has reached the self-sustaining rate, and then then they all decay to zero in the absence of input. We could easily avoid this problem, however, by choosing a smaller value for w^{ei2}, the feedback excitatory-inhibitory connection weight, that continues to satisfy the left inequality in condition A.16. The parameter set that we used in the simulations of the rate based model is given in Table A.1. $Uniform(min, max)$ indicates a uniform distribution on the interval $[min, max]$.

w^{ee}	1.9
w^{ei1}	0.7
w^{ei2}	0.3
w^{ie}	1.5
$w^{lateral}$	0.3
T_e	4
T_i	9
τ_e	0.04
τ_i	0.01
$w^f_{i,j,k}$ used in Figs. 2.3,2.4,2.6	independently sampled from Uniform[0.095,0.105]
$w^f_{i,j,k}$ used in Fig. 2.5	independently sampled from Uniform[0.045,0.055]
Parameters for the plastic connections described by Eq. 2.3	
w_{min}	0.08
w_{max}	0.12
v_{th}	8
τ_w	1
K	0.01

Table A.1: Parameters of the rate based model.

B

Details of the Spiking Implementation of Asymmetrically Coupled WTA Circuits

The models and the parameter values used in the spiking network closely follow those presented in ref. (Wang, 2002). The three types of synaptic currents in the neuron equation, Eq. 2.4, are modeled as:

$$I_{AMPA}(t) = (V_E - V_m(t)) \sum_i w^i_{AMPA} s^i_{AMPA}(t) \tag{B.1}$$

$$I_{NMDA}(t) = \frac{(V_E - V_m(t))}{1 + exp(-62 * V_m(t))/3.57} \sum_i w^i_{NMDA} s^i_{NMDA}(t) \tag{B.2}$$

$$I_{GABA}(t) = (V_I - V_m(t)) \sum_i w^i_{GABA} s^i_{GABA}(t) \tag{B.3}$$

V_E and V_I are the reversal potentials for the excitatory and inhibitory synapses respectively. w^i_{AMPA}, w^i_{NMDA}, and w^i_{GABA} control the magnitude of the changes in synaptic conductances in the the synapses formed by pre-synaptic neuron i when neuron i spikes (The first two are zero if i is an inhibitory neuron; the third is zero if i is an excitatory neuron). Eq. B.2 models the voltage dependent Magnesium blockage of the NMDA receptor-activated ion channel (Jahr & Stevens, 1990). The extra-cellular Magnesium concentration is assumed to be 1mM. $s^i_{AMPA}(t)$, $s^i_{NMDA}(t)$, and $s^i_{GABA}(t)$ reflect the time course of the change in synaptic conductances in response to spikes from neuron i:

$$\dot{s}^i_{AMPA}(t) = -\frac{s^i_{AMPA}(t)}{\tau_{AMPA}} + \sum_k \delta(t - t^i_k) \tag{B.4}$$

$$\dot{s}^i_{NMDA}(t) = -\frac{s^i_{NMDA}(t)}{\tau^{fall}_{NMDA}} + \frac{x^i(t)}{\tau^{rise}_{NMDA}}(1 - s^i_{NMDA}(t)) \tag{B.5a}$$

$$\dot{x}(t) = -\frac{x(t)}{\tau^{rise}_{NMDA}} + \sum_k \delta(t - t^i_k) \tag{B.5b}$$

$$\dot{s}^i_{GABA}(t) = -\frac{s^i_{GABA}(t)}{\tau_{GABA}} + \sum_k \delta(t - t^i_k) \tag{B.6}$$

where t^i_k is the k^{th} spike emitted by neuron i. The τ_{AMPA}, τ^{fall}_{NMDA}, τ^{rise}_{NMDA}, and τ_{GABA} time constants control the time course of conductance changes. The rise times of the $GABA$ and $AMPA$ mediated conductances are neglected.

The membrane capacitance, leak conductance and refractory period of an excitatory neuron are denoted by C^{exc}_m, g^{exc}_l, and T^{exc}_{ref} respectively and those of an inhibitory neuron by C^{inh}_m,

g_l^{inh}, and T_{ref}^{inh}. Excitatory and inhibitory neurons share the same firing threshold, V_{firing}, and resting/reset potential, V_l. Each excitatory population and each inhibitory population in Fig. 2.2 is made up of 30 neurons. Each neuron is described by Eq. 2.4. An arrow in Fig. 2.2 represents all-to-all connectivity from neurons in the source population to neurons in the target population. All connections originating from excitatory neurons give rise to excitatory post-synaptic currents mediated by both $NMDA$ and $AMPA$ receptors. All connections originating from inhibitory neurons give rise to inhibitory post-synaptic currents mediated by $GABA_A$ receptors. The magnitude of the changes in the synaptic conductances due to a pre-synaptic spike (the w terms in Eqs. B.1,B.2, and B.3) is drawn from random distributions with mean w_{AMPA}^{ee} and w_{NMDA}^{ee} for the recurrent excitatory synapses, w_{GABA}^{ie} for the inhibitory-excitatory synapses, w_{AMPA}^{ei1} and w_{NMDA}^{ei1} for the intra-stage excitatory-inhibitory synapses, w_{AMPA}^{ei2} and w_{NMDA}^{ei2} for the inter-stage feedback excitatory-inhibitory synapses, w_{AMPA}^{f} and w_{NMDA}^{f} for the inter stage feed-forward excitatory-excitatory synapses, and w_{AMPA}^{lat} and w_{NMDA}^{lat} for the lateral excitatory-excitatory synapses between neurons in different populations within each stage. The magnitude of conductance changes (the weights) are chosen independently for each synapse at the beginning of the simulation and kept fixed during the simulation. External excitation used to initiate, trigger, or steer the sequential activity takes the form of a Poisson spike train that activates $AMPA$ receptor-mediated conductances in neurons in the target population. The magnitude of the jump in the $AMPA$ conductance of each neuron in the target population due to an external spike is drawn from a Gaussian distribution with mean $w_{AMPA}^{external}$. The parameters of the spiking model are given in table B.1.

For the simulation in Fig. 2.10, we used the synaptic plasticity model proposed by (Graupner & Brunel, 2012) to modulate the magnitudes of the feed-forward $NMDA$ receptor mediated conductances, w_{NMDA}^{f}. For convenience, we reproduce the model below. Synaptic plasticity is governed by the calcium concentration $C(t)$ in the synapse:

$$\dot{C}(t) = \frac{-C(t)}{\tau_{CA}} + C_{pre}\sum_k \delta(t - t_k^{pre}) + C_{post}\sum_k \delta(t - t_k^{post}) \tag{B.7}$$

τ_{CA} is the time constant of calcium concentration decay, t_k^{pre} and t_k^{post} are the times of the k^{th} pre-synaptic spike and k^{th} postsynaptic spike respectively, and C_{pre} and C_{post} are the jumps in calcium concentration due to pre-synaptic spikes and post-synaptic spikes respectively. The synaptic efficacy, $\rho(t)$, varies in the range $[0, 1]$ according to:

$$\begin{aligned}\tau_\rho \dot{\rho}(t) = -\rho(t)(1-\rho(t))(0.5-\rho(t)) + \gamma_p(1-\rho(t))\Theta(C(t)-\theta_p) \\ - \gamma_d \rho(t)\Theta(C(t)-\theta_d)\end{aligned} \tag{B.8}$$

τ_ρ is the time constant of efficacy changes. $\Theta(x)$ is the Heaviside step function, γ_p and γ_d are the potentiation and depression rates respectively, and θ_p and θ_d are the calcium concentration thresholds for inducing potentiation and depression respectively. In the absence of pre- and post-synaptic activity, the efficacy, $\rho(t)$, drifts to one of the two fixed points: 0 or 1. We use $\rho(t)$ to modulate the magnitude of each $NMDA$ mediated conductance in the feed-forward path (from a neuron in an excitatory population in one stage to a neuron in an excitatory population in the subsequent stage). The conductance magnitude is modulated between two values, w_{min} and w_{max}:

$$w_{NMDA}^{f}(t) = \rho(t)(w_{max} - w_{min}) + w_{min} \tag{B.9}$$

Spiking network parameter	Mean value	$\frac{\text{mean}}{\text{std. deviation}}$
C_m^{exc}	0.5nF	0.0
C_m^{inh}	0.2nF	0.0
g_l^{exc}	25nS	0.0
g_l^{inh}	20nS	0.0
T_{ref}^{exc}	2ms	0.0
T_{ref}^{inh}	1ms	0.0
V_{firing}	-50mV	0.0
V_l	-70mV	0.0
V_E	0V	0.0
V_I	-70mV	0.0
τ_{AMPA}	2ms	0.0
τ_{NMDA}^{fall}	100ms	0.0
τ_{NMDA}^{rise}	2ms	0.0
τ_{GABA}	5ms	0.0
$w_{AMPA}^{external}$(Gaussian)	$4.0 * 10^{-9}$	2.5
w_{AMPA}^{ee}(Gaussian)	$1.0 * 10^{-10}$	2.5
w_{NMDA}^{ee}(Gaussian)	$8.0 * 10^{9}$	2.5
w_{GABA}^{ie}(Gaussian)	$2.0 * 10^{-9}$	2.5
w_{AMPA}^{ei1}(Gaussian)	$1.0 * 10^{-9}$	2.5
w_{NMDA}^{ei1}(Gaussian)	$4.0 * 10^{-9}$	2.5
w_{AMPA}^{ei2}(Gaussian)	$3.0 * 10^{-9}$	2.5
w_{NMDA}^{ei2}(Gaussian)	$7.0 * 10^{-9}$	2.5
w_{AMPA}^{f}(Gaussian)	$1.0 * 10^{-10}$	2.5
w_{NMDA}^{f}(Gaussian) in Fig. 2.8	$2.9 * 10^{-9}$	2.5
w_{NMDA}^{f}(Gaussian) in Fig. 2.9	$2.0 * 10^{-9}$	2.5
w_{AMPA}^{lat}(Gaussian)	$1.0 * 10^{-11}$	2.5
w_{NMDA}^{lat}(Gaussian)	$1.0 * 10^{-10}$	2.5

Parameters for the plastic NMDA conductance described by Eqs. B.7,B.8,B.9 and used in Fig. 2.10

τ_ρ	$100ms$
τ_{CA}	$20ms$
C_{pre}	1
C_{post}	2
γ_p	5
γ_d	1
θ_p	1.5
θ_d	0.5
w_{min}	$3 * 10^{-9}$
w_{max}	$3.4 * 10^{-9}$

Table B.1: Parameters of the spiking model.

C

Full Description of Coupled WTA Networks with Oscillatory Inhibition

The basic building block of all networks is the linear threshold unit (LTU). LTUs together with an oscillatory inhibitory population make up an oscillatory WTA as shown in Figs. 4.1a and 4.1b. The excitatory populations in different WTA circuits are coupled together to implement consistency conditions as shown in Fig. 4.1d. The full description of a network having M WTA circuits, with N_k excitatory population in WTA k is:

$$\tau^E \frac{d}{dt} x_{ij}^E(t) + x_{ij}^E(t) = f(w_{rec}^{\text{AMPA}} x_{ij}^E(t) + w_{rec}^{\text{NMDA}} s_{ij}^E(t) - w_{IE}^{\text{GABA}} x_i^I - w_{osc}^E H_i(t) + \sigma^{\text{LTU}} \eta_{ij}^E(t) + \sum_{\substack{k=1 \\ k \neq i}}^{M} \sum_{p=1}^{N_k} (w_{k,p \to i,j}^{\text{AMPA}} x_{kp}^E(t) + w_{k,p \to i,j}^{\text{NMDA}}(t) s_{kp}^E(t)) - T^E) \quad \text{(C.1a)}$$

$$\tau^I \frac{d}{dt} x_i^I(t) + x_i^I(t) = f(-w_{osc}^I H_i(t) + \sigma^{\text{LTU}} \eta_i^I(t) + \sum_{j=1}^{N_i} (w_{EI}^{\text{AMPA}} x_{ij}^E(t) + w_{EI}^{\text{NMDA}} s_{ij}^E(t)) - T^I) \quad \text{(C.1b)}$$

$$\tau^{\text{NMDA}} \frac{d}{dt} s_{ij}^E(t) + s_{ij}^E(t) = x_{ij}^E(t) \quad \text{(C.1c)}$$

$$\frac{d}{dt} \phi_i = \Omega_i + \sigma^{\text{OSC}} \kappa_i \delta(\phi_i) + \sum_{\substack{k=1 \\ k \neq i}}^{M} Z_{k \to i} sin(\phi_k(t) - \phi_i(t)) \quad \text{(C.1d)}$$

$$H_i(t) = \begin{cases} A & \text{if} \quad \phi_i < 2\pi d \\ 0 & \text{otherwise} \end{cases} \quad \text{(C.1e)}$$

$x_{ij}^E(t)$ and $x_i^I(t)$ are the the activities of the j^{th} excitatory population, and the inhibitory population in WTA i respectively. $f(x)$ is the linear threshold function: $f(x) = \max(0, x)$. ϕ_i is the phase of the oscillatory inhibition in WTA i. ϕ_i automatically wraps around to stay in the range $[0, 2\pi)$. $H_i(t)$ is the actual oscillatory inhibition waveform in WTA i, which, within each period, is high (with amplitude A) for a fraction $d < 1$ of the oscillation cycle. $x_{ij}^E(t)$ is low-pass filtered with time constant τ^{NMDA} to yield $s_{ij}^E(t)$ which is used to model slowly decaying currents mediated by NMDA receptors. τ^E, T^E and τ^I, T^I are the time constants and thresholds of the excitatory and inhibitory populations respectively. $w_{k,p \to i,j}^{\text{AMPA}}(t)$ and $w_{k,p \to i,j}^{\text{NMDA}}(t)$ are the weights of the inter-WTA AMPA- and NMDA-mediated connections (respectively) which connect population x_{kp}^E to population x_{ij}^E. These inter-WTA coupling weights are fixed in all simulations except the simulations of the perceptual multi-stability model where they are plastic and obey

the plasticity rule given in Eq. 4.2. The remaining weights are the intra WTA weights. $\eta_{ij}^{E}(t)$, $\eta_{i}^{I}(t)$ are informally treated as derivatives of independent, zero mean, unity variance Wiener processes. We could thus use the Euler-Maruyama method to carry out the noise simulations where in each time step Δt, the random processes $\eta_{ij}^{E}(t)$ and $\eta_{i}^{I}(t)$ generate independent Gaussian increments with zero mean and variance Δt. κ_i is a zero mean, unity variance Gaussian random variable. The term $\sigma^{\text{OSC}}\kappa_i\delta(\phi_i)$ in Eq. C.1d indicates that at the beginning of each oscillation cycle, $\phi_i = 0$, a random increment $\sigma^{\text{OSC}}\kappa_i$ is added to the oscillation phase. This increment is only added once per cycle so the phase has to cross the 2π point and reset before another increment is added. σ^{LTU} and σ^{OSC} are noise scaling factors. $Z_{k\to i}$ is the phase coupling strength between oscillatory inhibition in WTAs k and i. We always use symmetric phase coupling: $Z_{i\to j} = Z_{j\to i}$. The phase coupling model follows the Kuramoto model (Strogatz, 2000).

Table C.1 contains the parameter values for the network simulations in Figs. 4.1, 4.3, 4.4, 4.5 and E.1. The noise and phase coupling terms are only non-zero in the simulations in Fig. 4.4. Table C.2 contains the parameter values used in the Sudoku network in Fig. 4.2. Table C.3 contains the parameter values used in the simulations of the perceptual multi-stability model (Figs. 4.6 and 4.7). Parameter values are only shown in Tables C.2 and C.3 if they differ from those in Table C.1.

Oscillatory Inhibition		
d	0.6	Fraction of oscillation cycle with active inhibitory output
$p(f)$	Uniform(45,46) Hz	Distribution of oscillation frequencies
A	40 Hz	Amplitude of oscillatory inhibition
w_{osc}^{E}	-3	Oscillatory inhibition to excitatory populations weight
w_{osc}^{I}	-1	Oscillatory inhibition to inhibitory populations weight
Neuron Populations		
τ^{E}	0.3 ms	Excitatory population time constant
τ^{I}	0.2 ms	Inhibitory population time constant
τ^{NMDA}	80 ms	NMDA time constant
T^{I}	8	Inhibitory population threshold
T^{E}	-18	Excitatory population threshold
w_{rec}^{AMPA}	1.2	Recurrent intra-WTA non-NMDA excitatory weight
w_{rec}^{NMDA}	0.004	Recurrent intra-WTA NMDA excitatory weight
w_{EI}^{AMPA}	0.5	Intra-WTA non-NMDA excitatory to inhibitory weight
w_{EI}^{NMDA}	0.3	Intra-WTA NMDA excitatory to inhibitory weight
w_{IE}^{GABA}	-1	Intra-WTA inhibitory to excitatory weight
$w_{k,p\to i,j}^{\text{AMPA}}$	0.06	Weight of inter-WTA non-NMDA coupling connection from $x_{k,p}^{E}$ to $x_{i,j}^{E}$
$w_{k,p\to i,j}^{\text{NMDA}}$	0.005	Weight of inter-WTA NMDA coupling connection from $x_{k,p}^{E}$ to $x_{i,j}^{E}$
Noise and Phase Coupling Parameters (Only for the Simulations in Fig. 4)		
σ^{LTU}	200	Noise scaling factor in the high noise case
σ^{LTU}	50	Noise scaling factor in the low noise case
σ^{OSC}	0.7	Phase perturbation per cycle for both the low-noise and high-noise case
$Z_{0\to3}$,$Z_{3\to0}$	5	Phase coupling strength between V0-V3

Table C.1: Network parameters used for the simulations in Figs. 4.1, 4.3, 4.4, 4.5 and E.1.

Oscillatory Inhibition		
d	0.17	Fraction of oscillation cycle with active inhibitory output
$p(f)$	Uniform(40,60) Hz	Distribution of oscillation frequencies
Neuron Populations		
τ^E	0.5 ms	Excitatory population time constant
T^I	6	Inhibitory population threshold
T^E	-2	Excitatory population threshold
w_{rec}^{AMPA}	1.8	Recurrent intra-WTA non-NMDA excitatory weight
w_{rec}^{NMDA}	0.0001	Recurrent intra-WTA NMDA excitatory weight
w_{EI}^{AMPA}	0.6	Intra-WTA non-NMDA excitatory to inhibitory weight
w_{IE}^{GABA}	-1.6	Intra-WTA inhibitory to excitatory weight
$w_{k,p\to i,j}^{\text{AMPA}}$	0.002	Weight of inter-WTA non-NMDA coupling connection from $x_{k,p}^E$ to $x_{i,j}^E$
$w_{k,p\to i,j}^{\text{NMDA}}$	0.0	Weight of inter-WTA NMDA coupling connection from $x_{k,p}^E$ to $x_{i,j}^E$

Table C.2: Network parameters used for the simulations of the Sudoku network in Fig. 4.2. Only parameter values that are different from those in Table C.1 are shown.

Oscillatory Inhibition		
d	0.27	Fraction of oscillation cycle with active inhibitory output
f_G	45 Hz	Oscillation frequency to which oscillators couple when synchronized
$p(f)$	Uniform(43,53) Hz	Distribution of oscillation frequencies
Neuron Populations		
τ^E	0.5 ms	Excitatory population time constant
T^E	-2	Excitatory population threshold
w_{rec}^{NMDA}	0.005	Recurrent intra-WTA NMDA excitatory weight
w_{IE}^{GABA}	-1.1	Intra-WTA inhibitory to excitatory weight
Plasticity Rule Parameters		
w_{max}	0.03	Maximal weight of plastic synapse
w_{min}	0	Minimal weight of plastic synapse
θ	38 Hz	Learning threshold
η_{up}	7.5×10^{-5} /s	Potentiation rate
η_{down}	3.75×10^{-5} /s	Depression rate
d_{up}	0.002 /s	Upward drift rate
d_{down}	0.0005 /s	Downward drift rate
Readout WTA		
w_{in}	0.000216	Weight from hidden WTA to readout WTA
w_{rec}^{AMPA}	1	Recurrent intra-WTA non-NMDA excitatory weight
w_{rec}^{NMDA}	0.001	Recurrent intra-WTA NMDA excitatory weight
w_{EI}^{AMPA}	0.3	Intra-WTA non-NMDA excitatory to inhibitory weight

Table C.3: Network parameters used in the perceptual multi-stability model (Figs. 4.6 and 4.7). Only parameter values that are different from those in Table C.1 are shown.

D

Proofs of Propositions 1 and 2 in Chapter 4

The frequencies of the local oscillatory inhibition in the WTA circuits are different and are not rational multiples of each other. Having one oscillation frequency that is a rational multiple of another is statistically impossible in a physical system with real-valued frequencies. Let $\mathbf{\Phi}(t) = [\phi_1(t), \phi_2(t), \ldots, \phi_n(t)]$ be the vector of the phases of the oscillatory inhibition in the n WTA circuits in the network at time t where $\phi_i \in [0, 2\pi)$. By virtue of the incommensurability of the oscillatory inhibition frequencies, $\mathbf{\Phi}(t)$ follows a quasi-periodic trajectory. For any initial phase vector $\mathbf{\Phi}(0)$, and any $\hat{\mathbf{\Phi}} \in [0, 2\pi]^n$ and $\epsilon > 0$, there is a time $\hat{t}$ such that $|\mathbf{\Phi}(\hat{t}) - \hat{\mathbf{\Phi}}| < \epsilon$. The vector of phases $\mathbf{\Phi}(t)$ thus densely fills up the n-dimensional hyper-torus of all possible phases, which implies that $\mathbf{\Phi}(t)$ cannot have a periodic trajectory. Since it is the phase relations between the WTA circuits that determine the effectiveness of the different consistency conditions, the ergodicity of the phase vector $\mathbf{\Phi}(t)$ ensures that all possible combinations of the strengths of the consistency conditions are eventually explored. In a WTA circuit W_i, the oscillatory inhibition is on for h_i seconds and off for l_i seconds during each cycle. In order to prove some useful results about the trajectory of $d(t)$, we require that $h_i > l_j \forall i, j \neq i$. This condition, together with the ergodicity of the phase vector, $\mathbf{\Phi}$, guarantees that for any WTA circuit W_r, there is bound to come an oscillation cycle where the winner selection is unaffected by the activity in any other WTA, because during that cycle, oscillatory inhibition switches off in W_r, W_r selects a winner, and gets shut down again by oscillatory inhibition within l_r seconds, while all other WTA circuits are quiescent.

Proposition 1. *The only fixed points of $d(t)$ are the configurations that satisfy all consistency conditions.*

Proof. Assume $d(t)$ has a fixed point which encodes a particular non-changing configuration and that this configuration violates one or more consistency conditions. Let V_1 and V_2 be two WTA circuits that are part of a violated consistency condition. Due to the ergodicity of the phase vector, $\mathbf{\Phi}(t)$, there is bound to come an oscillation cycle for V_2 which has the following properties: during the part of the cycle when oscillatory inhibition is off (low) in V_2, oscillatory inhibition is on (high) in all other WTA circuits except V_1 and the peak of activity in the excitatory populations in V_1 occurs at the time in which it can dictate the winner in V_2. The influence of V_1 on V_2 is thus unopposed by any other WTA and this influence acts to enforce the violated consistency condition by changing the identity of the winner in V_2. Thus, a configuration that violates one or more consistency conditions is unstable. Assume $d(t)$ has reached a point $\bar{d}$ which encodes a configuration that satisfies all consistency conditions. The currently winning population in each WTA circuit will receive from other WTAs either a stronger or an equal input as compared to the other populations in the WTA. In both cases, and due to the hysteresis mechanism mediated by the slow recurrent NMDA current, the winning population

in each WTA circuit keeps on winning and the global configuration $\bar{d}$ remains unchanged hence $\bar{d}$ is a fixed point. □

Proposition 2. *$d(t)$ is not periodic as long as it has not reached a fixed point.*

Proof. Since $d(t)$ has not reached a fixed point, $d(t)$ alternates between a number of configurations. Let V be a WTA in which the identity of the winning population changes between these configurations. Assume $d(t)$ has period T. Fix t_0, at some t_1 where $t_0 \leq t_1 < t_0 + T$, WTA V changes its state, i.e. it selects a winning population that is different from the winning population in the previous cycle. This change is reflected in $d(t)$ at t_1 when the oscillatory inhibition switches on in V. V has to change its state in the same way at $t_1 + nT$ where $n = 1, 2, \ldots$ by the periodicity assumption of $d(t)$. So T has to be a multiple of the oscillation period of V. Since the frequencies of oscillatory inhibition in the WTA circuits are incommensurable, T is incommensurable with the oscillation periods of all WTA circuits except V. Hence, there is bound to come an oscillation cycle for V with the following two properties: during the part of the cycle when oscillatory inhibition is off in V, oscillatory inhibition is on in all other WTA circuits; and the cycle terminates at $t_1 + kT$ (where k is an integer) when the inhibition in V goes high. So it is impossible for V to select a winner in this cycle that is different from the winner it selected in the previous cycle (in the absence of external influence throughout this cycle, the hysteresis mechanism mediated by the slow NMDA recurrent connections yields the same winner as the previous cycle). This contradicts the initial assumptions about V (that it selects a different winner at $t_1 + kT$) and establishes the aperiodicity of $d(t)$. □

E

Interpretation of the Dynamics of Oscillatory Coupled WTA Networks as Markov Chain Monte Carlo Sampling

There are multiple MCMC sampling operators that can approximate the dynamics of the oscillatory WTA networks in chapter 4. The operator described in this section was developed by Lorenz Muller and strikes a balance between complexity and accuracy in approximating the network dynamics. It is based on the simplifying assumptions of instantaneous WTA functionality, and randomized (rather than oscillatory) node update ordering. In MCMC sampling, a stochastic Markovian transition operator R defined over a space $\mathbf{X}$ is used to generate a stochastic sequence of points, or samples, in $\mathbf{X}$: $x_1, x_2, \ldots$, where x_{n+1} is a sample drawn from the conditional probability distribution:

$$p(x_{n+1} = z | x_n = y) = R(y \to z) \tag{E.1}$$

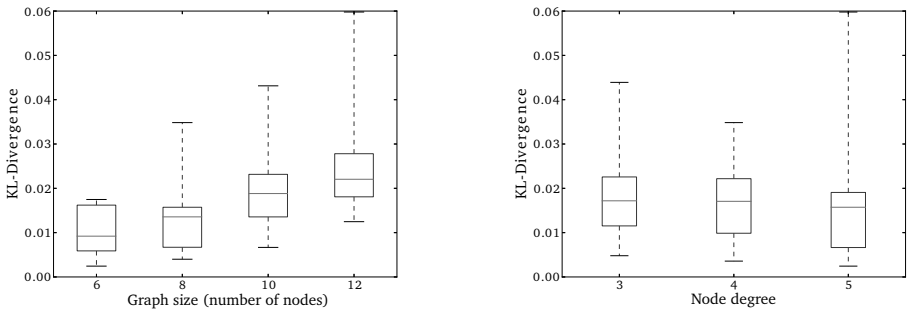
$R(y \to z)$ is the probability that the next point in the sequence is z given that the current point is y. If R is irreducible (for any two states, x_1 and x_2, x_2 will with non-zero probability appear in the random sequence started from x_1 after a finite number of steps), and R is aperiodic (starting from any state x, the indices of the positions in the sequence where x has a non-zero probability of occurring do not have a common divisor other than 1) then for any starting point x_0, $p(x_n = y)$ approaches the quantity $q(y)$ as n gets large for all $y \in \mathbf{X}$. In other words, the points $x_n, x_{n+1}, \ldots$ are samples drawn from the probability distribution $q(x)$, called the stationary or invariant distribution of the transition operator R, for large enough n.

Given a network of interacting WTA circuits, our goal is to find a Markovian stochastic transition operator, T, that is defined over the space of all network configurations and which approximates the way the network trajectory moves through this configuration space. We describe here how, given a particular network configuration x_n, the next configuration x_{n+1} is stochastically evaluated. This evaluation procedure implicitly defines the transition operator T and it attempts to approximate the way the configuration of the actual network changes.

1. Choose at random a WTA V to be updated. This corresponds to a WTA in which oscillatory inhibition has just switched off and is now selecting a winner.

2. Choose at random a subset of the consistency conditions that involve the WTA V. Denote this subset as C. The WTA circuits involved in this subset are assumed to have the proper phase relations to V that enable them to affect the winner selection in V.

3. For each possible state s_i of WTA V (the number of possible states is the number of excitatory populations in the WTA), attach a number n_i that counts the number of consistency conditions in C that are satisfied if V is in state s_i. The states of the other WTA circuits



(a) (b)

Figure E.1: The KL-Divergence (in nat) between the 'probability distribution' induced by the network trajectory and the stationary distribution of the equivalent MCMC sampler, calculated for 60 randomly generated regular graphs with sizes (number of nodes) in $\{6, 8, 10, 12\}$ and node degrees in $\{3, 4, 5\}$. There are five graph instances for each size-degree combination. In (a) the KL-Divergence is plotted as a function of the number of nodes in the graph (15 data points per item) and in (b), it is plotted as a function of degree (20 data points per item). The horizontal line within each box indicates the median, the box outlines the 1st and 3rd quartiles and the whiskers show the full range of the data.

are fixed and encoded in x_n. Find the maximum of n_i across all states s_i and denote it as n. Define $S_C = \{s_k : n_k = n\}$. S_C is thus the set of possible states for WTA V that are favored by the majority of the consistency conditions in C. If the current state of V is in S_C, then the state of V remains unchanged and $x_{n+1} = x_n$. This reflects the hysteresis mechanism mediated by the slow NMDA current that favors the WTA population that won in the previous inhibition cycle. If the current state of V is not in S_C, then a new state for V is selected randomly from S_C and x_{n+1} is updated accordingly. Note that at most one WTA changes its state between x_n and x_{n+1}.

The stationary probability distribution of this MCMC operator does not exactly reproduce the relative durations of the different configurations in the actual network as shown in Fig. 4.3. There are many reasons for this discrepancy:

- The state of a WTA (the identity of the winning population) in one oscillation cycle is not only affected by its state in the preceding cycle but also by its state in cycles further in the past. That is because the time constant of the slow recurrent NMDA current is 80 ms and the mean period of the oscillatory inhibition in the WTA circuits is 22 ms. When a population wins in one inhibition cycle, its increased recurrent NMDA current takes many cycles to decay, which might affect the winner selection process many cycles later. The Markovity property which is fundamental to the construction of T is thus not exact.

- Some WTA circuits (those that have a higher oscillation frequency) update more often than others. This violates the assumption inherent in step 1 in the MCMC update scheme, that WTA circuits update equally often on average. In sampling algorithms like Gibbs sampling, differences in the variables' update frequencies do not affect the stationary distribution, but in our case they do. One can show that this is a consequence of the fact that the transition operator T does not obey the detailed balance equation, while the transition operator for the Gibbs sampler does.

- In comparing the statistics of $d(t)$ and those of the sequence generated by T, we are comparing the durations of the different configurations in $d(t)$ to their relative frequencies of occurrence in the MCMC sequence. Thus, we are in essence assigning an equal duration to each MCMC sample, which exposes another assumption of the MCMC scheme, namely, that the WTA circuits update their states at equal intervals (the updated state might be the same as the previous one). In the actual network, there is no clear-cut update cycle. The WTA circuits might update their states in very quick succession so the intermediate configurations might have very short durations, or one configuration might persist for a relatively long time before a WTA updates its state (i.e. the inhibition goes high in this WTA and the winner is evaluated).

To further investigate the difference between the network statistics and those of the MCMC operator, we constructed random regular graphs of different sizes and node degrees. From each graph we constructed a network in the following way: each node in the graph is a binary WTA and each edge represents a consistency condition which could either be an equality or inequality condition. We then used Kullback-Leibler(KL) divergence to quantify the difference between the 'probability distribution' generated by the network and the stationary distribution of the equivalent MCMC operator. The results are shown in Fig. E.1 for different graph/network sizes and node degrees. The above-mentioned points of discrepancy lead to a more pronounced difference in the two sets of statistics in larger networks/graphs. To verify that this is not an effect of finite sample size, we generated a larger number of MCMC samples and ran the network for longer durations; the KL-divergence between the two sets of statistics remained virtually unchanged. The match between the two sets of statistics does not seem to depend on the network/graph degree.

Properties of the Network-approximating MCMC Operator

The transition operator T for an arbitrary network constructed as outlined above is aperiodic as there is always a non-zero probability that any configuration persists for an arbitrary number of steps in the sequence. Unlike transition operators used in typical MCMC applications, T does not obey detailed balance so the generated Markov chain of samples is irreversible. Also, T is not irreducible in general. This means that in general, for networks of interacting WTA circuits, the space of possible configurations can be decomposed into a number of non-communicating sets and one set of transient configurations. Starting the chain from a configuration from the transient set, there is a non-zero probability that this configuration will never be visited again. If started in one of the non-communicating sets, the network trajectory will always stay in this set of configurations. Hence, the long-term frequency of occurrence, or the 'probability distribution', over the network configurations will in general depend on the initial configuration of the network.

The reducibility of T is a desirable property in some circumstances. For example, T is reducible for a network where there is a configuration that satisfies all consistency conditions as this configuration persists forever. Having fully consistent configurations as absorbing states enables this kind of network to be used for solving constraint satisfaction problems. This behavior is useful in the perceptual multi-stability model as it allows the network to form stable percepts if the input is unambiguous. T is also reducible for a network where there is a configuration that violates all consistency conditions as this configuration will never be visited. This could be beneficial as it reduces the space of configurations explored by the network by discarding fully inconsistent states for which there is no evidence, either from external input, or from the consistency conditions in the network. More elaborate networks can be constructed that do not fall

into these two categories but that map to a reducible stochastic transition operator by virtue of having two or more non-communicating sets of recurrent states (i.e. non-transient states). External input can switch the network configuration from one subset to another. The network can thus explore different parts of the configuration space based on external input. The reducibility of T is, however, undesirable if the goal is to generate samples from a unique probability distribution irrespective of initial conditions, which is a typical requirement in many MCMC applications.

References

Aaronson, S. (2005). Guest column: Np-complete problems and physical reality. ACM Sigact News, 36, 30–52. 118

Abbott, L. & Regehr, W. (2004). Synaptic computation. Nature, 431, 796–803. 73

Abeles, M. et al. (1982). Local cortical circuits. (Springer, Berlin). 12

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. Cognitive science, 9, 147–169. 6, 56

Afraimovich, V., Zhigulin, V., & Rabinovich, M. (2004). On the origin of reproducible sequential activity in neural circuits. Chaos: An Interdisciplinary Journal of Nonlinear Science, 14, 1123–1129. 12

Ainsworth, M., Lee, S., Cunningham, M., Roopun, A., Traub, R., & Whittington, N. K. M. (2011). Dual gamma rhythm generators control interlaminar synchrony in auditory cortex. The Journal of Neuroscience, 31, 17040–17051. 44

Amit, D. (1988). Neural networks counting chimes. Proc.Natl.Acad.Sci, USA, 85, 2141–2145. 12

Amit, D. (1992). Modeling brain function: The world of attractor neural networks. (Cambridge University Press). 31

Amit, D. & Brunel, N. (1997). Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. Cerebral Cortex, 7, 237–252. 31, 45

Axmacher, N., Mormann, F., Fernández, G., Elger, C., & Fell, J. (2006). Memory formation by neuronal synchronization. Brain research reviews, 52, 170–182. 61

Balint, A. & Schöning, U. (2012). Choosing probability distributions for stochastic local search and the role of make versus break. In Theory and Applications of Satisfiability Testing–SAT 2012. (Springer), pp. 16–29. 90

Barahona, F. (1982). On the computational complexity of ising spin glass models. Journal of Physics A: Mathematical and General, 15, 3241. 4, 104, 119

Belov, A., Diepol, D., Heule, M., & Järvisalo, M. (2014). Sat competition. 90

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J., Alvarez-Icaza, R., Arthur, J., Merolla, P., & Boahen, K. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. Proceedings of the IEEE, 102, 699–716. 7, 73

Benzi, R., Sutera, A., & Vulpiani, A. (1981). The mechanism of stochastic resonance. Journal of Physics A: mathematical and general, 14, L453. 112

Berkes, P., Orbán, G., Lengyel, M., & Fiser, J. (2011). Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. Science, 331, 83–87. 33, 44, 59

Bi, G.-Q. & Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. Journal of Neuroscience, 18, 10464–10472. 73

Bienenstock, E., Cooper, L., & Munro, P. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. Jour. Neurosci., 2, 32–48. 21, 35, 56

Binzegger, T., Douglas, R., & Martin, K. (2004). A quantitative map of the circuit of cat primary visual cortex. Journal of Neuroscience, 24, 8441–53. 12, 13, 14, 73

Boahen, K. (2000). Point-to-point connectivity between neuromorphic chips using address-events. IEEE Transactions on Circuits and Systems II, 47, 416–34. 66, 98

Bosman, C., Schoffelen, J.-M., Brunet, N., Oostenveld, R., Bastos, A., Womelsdorf, T., Rubehn, B., Stieglitz, T., De Weerd, P., & Fries, P. (2012). Attentional stimulus selection through selective synchronization between monkey visual areas. Neuron, 75, 875–888. 43, 46, 52, 54, 61

Bostrom, N. (2003). Are we living in a computer simulation? The Philosophical Quarterly, 53, 243–255. 118

Brader, J., Senn, W., & Fusi, S. (2007). Learning real world stimuli in a neural network with spike-driven synaptic dynamics. Neural Computation, 19, 2881–2912. 65, 74, 75, 76, 83, 85

Braunstein, A., Mézard, M., & Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. Random Structures & Algorithms, 27, 201–226. 119

Brosch, M., Budinger, E., & Scheich, H. (2002). Stimulus-related gamma oscillations in primate auditory cortex. Journal of Neurophysiology, 87, 2715–2725. 120

Brunel, N. & Wang, X. J. (2003). What determines the frequency of fast network oscillations with irregular neural discharges? I. synaptic dynamics and excitation-inhibition balance. Journal of Neurophysiology, 90, 415–430. 43, 120

Buesing, L., Bill, J., Nessler, B., & Maass, W. (2011). Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. PLoS computational biology, 7, e1002211. 52, 107, 112

Burns, S., Samuel, P., Xing, D., Shelley, M., & Shapley, R. (2010). Searching for autocoherence in the cortical network with a time-frequency analysis of the local field potential. The Journal of Neuroscience, 30, 4033–4047. 45

Buzsáki, G. & Draguhn, A. (2004). Neuronal oscillations in cortical networks. Science, 304, 1926–1929. 43, 44

Buzsáki, G. & Wang, X.-J. (2012). Mechanisms of gamma oscillations. Annual review of neuroscience, 35, 203–225. 43, 44

Cardin, J., Carlén, M., Meletis, K., Knoblich, U., Zhang, F., Deisseroth, K., Tsai, L.-H., & Moore, C. (2009). Driving fast-spiking cells induces gamma rhythm and controls sensory responses. Nature, 459, 663–667. 43, 45

Chen, L. & Aihara, K. (1995). Chaotic simulated annealing by a neural network model with transient chaos. Neural networks, 8, 915–930. 34

Chicca, E., Stefanini, F., Bartolozzi, C., & Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. Proceedings of the IEEE, 102, 1367–1388. 76

Chicca, E., Whatley, A., Lichtsteiner, P., Dante, V., Delbruck, T., Del Giudice, P., Douglas, R., & Indiveri, G. (2007). A multi-chip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity. IEEE Transactions on Circuits and Systems I, 5, 981–993. 66

Chua, L. (1971). Memristor-the missing circuit element. Circuit Theory, IEEE Transactions on, 18, 507–519. 73

Clopath, C. & Gerstner, W. (2010). Voltage and spike timing interact in STDP – a unified model. Frontiers in Synaptic Neuroscience, 2, 1–11. 83

Copeland, B. (1996). What is computation? Synthese, 108, 335–359. 4

Crutchfield, J. (1994). Critical computation, phase transitions, and hierarchical learning. Towards the Harnessing of Chaos, Amsterdam. 4, 34

da Costa, N. & Martin, K. (2010). Whose cortical column would that be? Frontiers in neuroanatomy, 4. 13

de Almeida, L., Idiart, M., & Lisman, J. (2009). A second function of gamma frequency oscillations: an e%-max winner-take-all mechanism selects which cells fire. The Journal of Neuroscience, 29, 7497–7503. 60

Deco, G. & Rolls, E. (2003). Attention and working memory: a dynamical model of neuronal activity in the prefrontal cortex. European Journal of Neuroscience, 18, 2374–2390. 43

Deco, G. & Rolls, E. (2005). Sequential memory: a putative neural and synaptic dynamical mechanism. Journal of Cognitive Neuroscience, 17, 294–307. 12

Deiss, S., Douglas, R., & Whatley, A. (1998). A pulse-coded communications infrastructure for neuromorphic systems. In Pulsed Neural Networks, W. Maass & C. Bishop, eds. (MIT Press), chap. 6, pp. 157–78. 98

Delbruck, T. & Lichtsteiner, P. (2006). Fully programmable bias current generator with 24 bit resolution per bias. In Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, pp. 4–pp. IEEE. 79

der Heydt, R. V., Peterhans, E., & Baumgartner, G. (1984). Illusory contours and cortical neuron responses. Science, 224, 1260–1262. 121

Deutsch, D. (1985). Quantum theory, the church-turing principle and the universal quantum computer. In Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, no. 1818, pp. 97–117. The Royal Society. 118

Douglas, R. & Martin, K. (1992). A functional microcircuit for cat visual cortex. Jour. Physiol., 440, 735–769. 13, 34

Douglas, R. & Martin, K. (2004). Neural circuits of the neocortex. Annual Review of Neuroscience, 27, 419–51. 13, 14, 34, 43, 44

Douglas, R., Martin, K., & Whitteridge, D. (1989). A canonical microcircuit for neocortex. Neural Computation, 1, 480–488. 13

Ercsey-Ravasz, M. & Toroczkai, Z. (2011). Optimization hardness as transient chaos in an analog approach to constraint satisfaction. Nature Physics, 7, 966–970. 87

Ermentrout, B. (1992). Complex dynamics in winner-take-all neural nets with slow inhibition. Neural networks, 5, 415–431. 13

FACETS (2005–2009). Fast analog computing with emergent transient states in neural architectures (FACETS). FP6-2005-015879 EU Grant. 7

Fairhall, S. & Caramazza, A. (2013). Brain regions that represent amodal conceptual knowledge. The Journal of Neuroscience, 33, 10552–10558. 11

Fasnacht, D. & Indiveri, G. (2011). A PCI based high-fanout AER mapper with 2 GiB RAM look-up table, 0.8 μs latency and 66 MHz output event-rate. In Conference on Information Sciences and Systems, CISS 2011, pp. 1–6. Johns Hopkins University. 66, 104

Feldman, J. A. & Ballard, D. H. (1982). Connectionist models and their properties. Cognitive science, 6, 205–254. 2

Fell, J. & Axmacher, N. (2011). The role of phase synchronization in memory processes. Nature Reviews Neuroscience, 12, 105–118. 61

Fiete, I., Senn, W., Wang, C., & Hahnloser, R. (2010). Spike-time-dependent plasticity and heterosynaptic competition organize networks to produce long scale-free sequences of neural activity. Neuron, 65, 563–576. 12

Fodor, J. (1975). The language of thought, vol. 5. (Harvard University Press). 2, 25

Fodor, J. & Pylyshyn, Z. (1993). Connectionism and cognitive architecture. In Readings in philosophy and cognitive science, pp. 801–818. MIT Press. 2

Fourcaud, N. & Brunel, N. (2002). Dynamics of the firing probability of noisy integrate-and-fire neurons. Neural computation, 14, 2057–2110. 45

Freund, T., Martin, K., Smith, A., & Somogyi, P. (1983). Glutamate decarboxylase-immunoreactive terminals of golgi-impregnated axoaxonic cells and of presumed basket cells in synaptic contact with pyramidal neurons of the cat's visual cortex. Journal of Comparative Neurology, 221, 263–278. 13

Fries, P. (2005). A mechanism for cognitive dynamics: neuronal communication through neuronal coherence. Trends in cognitive sciences, 9, 474–480. 40, 43, 46, 54, 61

Fries, P. (2009). Neuronal gamma-band synchronization as a fundamental process in cortical computation. Annual review of neuroscience, 32, 209–224. 44, 45

Fries, P., Nikolić, D., & Singer, W. (2007). The gamma cycle. Trends in neurosciences, 30, 309–316. 60

Friston, K. (1994). Functional and effective connectivity in neuroimaging: a synthesis. Human Brain Mapping, 2, 56–78. 46

Friston, K. (2003). Learning and inference in the brain. Neural Networks, 16, 1325–1352. 2, 4, 44, 59

Fusi, S., Annunziato, M., Badoni, D., Salamon, A., & Amit, D. (2000). Spike–driven synaptic plasticity: theory, simulation, VLSI implementation. Neural Computation, 12, 2227–58. 74

Fusi, S. & Mattia, M. (1999). Collective behavior of networks with linear (VLSI) integrate and fire neurons. Neural Computation, 11, 633–52. 17, 45

Garey, M., Johnson, D., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of operations research, 1, 117–129. 3, 87

Gershman, S. J., Vul, E., & Tenenbaum, J. B. (2012). Multistability and perceptual inference. Neural computation, 24, 1–24. 52, 57

Gerstein, G. L. & Mandelbrot, B. (1964). Random walk models for the spike activity of a single neuron. Biophysical journal, 4, 41. 112

Gomez-Rodriguez, F., Paz, R., Linares-Barranco, A., Rivas, M., Miro, L., Vicente, S., Jimenez, G., & Civit, A. (2006). AER tools for communications and debugging. In International Symposium on Circuits and Systems, (ISCAS), 2006, pp. 3253–3256. IEEE. 66

Goodale, M. & Milner, A. (1992). Separate visual pathways for perception and action. Trends in neurosciences, 15, 20–25. 11

Graupner, M. & Brunel, N. (2012). Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. Proceedings of the National Academy of Sciences, 109, 3991–3996. 24, 83, 130

Gray, C. & Singer, W. (1989). Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. Proc. Natl. Acad. Sci., 86, 1698–1702. 120

Gregory, R. (1980). Perceptions as hypotheses. Philosophical Transactions of the Royal Society of London. B, Biological Sciences, 290, 181–197. 2, 4

Grosof, D., Shapley, R., & Hawken, M. (1993). Macaque vi neurons can signal illusory contours. Nature, 365. 121

Grover, L. (1996). A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212–219. ACM. 4

Habenschuss, S., Jonke, Z., & Maass, W. (2013). Stochastic computations in cortical microcircuit models. PLoS computational biology, 9, e1003311. 33, 49, 87

Hinton, G., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18, 1527–1554. 65, 107

Hinton, G. & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. Science, 313, 504–507. 65, 107

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. Neural computation, 14, 1771–1800. 56

Holt, G., Softky, W., Koch, C., & Douglas, R. (1996). Comparison of discharge variability in vitro and in vivo in cat visual cortex neurons. Journal of Neurophysiology, 75, 1806–1814. 107

Hoos, H. H. & Stützle, T. (1998). Satlib–the satisfiability library. Web site at: http://www. satlib. org. 102

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79, 2554–2558. 12, 31, 87

Hopfield, J. (1994). Physics, computation, and why biology looks so different. Journal of theoretical biology, 171, 53–60. 5

Hopfield, J. (2008). Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. Neural computation, 20, 1119–1164. 49

Hopfield, J. (2010). Neurodynamics of mental exploration. Proceedings of the National Academy of Sciences, 107, 1648–1653. 12

Hopfield, J. & Tank, D. (1985). neural computation of decisions in optimization problems. Biological cybernetics, 52, 141–152. 34, 87

Hopfield, J. & Tank, D. (1986). Computing with neural circuits- a model. Science, 233, 625–633. 87

Imam, N., Akopyan, F., Arthur, J., Merolla, P., Manohar, R., & Modha, D. (2012). A digital neurosynaptic core using event-driven qdi circuits. In Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on, pp. 25–32. 66

Indiveri, G., Chicca, E., & Douglas, R. (2006). A VLSI array of low-power spiking neurons and bistable synapses with spike–timing dependent plasticity. IEEE Transactions on Neural Networks, 17, 211–221. 99

Indiveri, G., Linares-Barranco, B., Hamilton, T., van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., Schemmel, J., Cauwenberghs, G., Arthur, J., Hynna, K., Folowosele, F., Saighi, S., Serrano-Gotarredona, T., Wijekoon, J., Wang, Y., & Boahen, K. (2011). Neuromorphic silicon neuron circuits. Frontiers in Neuroscience, 5, 1–23. 75

Indiveri, G., Linares-Barranco, B., Legenstein, R., Deligeorgis, G., & Prodromakis, T. (2013). Integration of nanoscale memristor synapses in neuromorphic computing architectures. Nanotechnology, 24, 384010. 73

Indiveri, G. & Liu, S.-C. (2015). Memory and information processing in neuromorphic systems. Proceedings of the IEEE, 103, 1379–1397. 105

Indiveri, G., Stefanini, F., & Chicca, E. (2010). Spike-based learning with a generalized integrate and fire silicon neuron. In International Symposium on Circuits and Systems, (ISCAS), 2010, pp. 1951–1954. IEEE, Paris, France. 75

Jadi, M. & Sejnowski, T. (2014). Regulating cortical oscillations in an inhibition-stabilized network. Proceedings of the IEEE, 102, 830–842. 43, 120

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach. (GMD-Forschungszentrum Informationstechnik). 6

Jahr, C. & Stevens, C. (1990). Voltage dependence of nmda-activated macroscopic conductances predicted by single-channel kinetics. The Journal of Neuroscience, 10, 3178–3182. 129

Jin, X., Lujan, M., Plana, L., Davies, S., Temple, S., & Furber, S. (2010). Modeling spiking neural networks on SpiNNaker. Computing in Science & Engineering, 12, 91–97. 66

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., & Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. Nano letters, 10, 1297–1301. 73, 74

Johnson, M., Amin, H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A., Johansson, J., Bunyk, P., et al. (2011). Quantum annealing with manufactured spins. Nature, 473, 194–198. 4, 119

Joshi, S., Deiss, S., Arnold, M., Yu, T., & Cauwenberghs, G. (2010). Scalable event routing in hierarchical neural array architecture with global synaptic connectivity. In Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on, pp. 1–6. IEEE. 104

Kamgar-Parsi, B. (1992). Dynamical stability and parameter selection in neural optimization. In Neural Networks, 1992. IJCNN., International Joint Conference on, vol. 4, pp. 566–571. IEEE. 34

Kanefsky, B. & Taylor, W. (1991). Where the really hard problems are. In Proceedings of IJCAI, vol. 91, pp. 163–169. 38

Kersten, D. & Yuille, A. (2003). Bayesian models of object perception. Current opinion in neurobiology, 13, 150–158. 55

Kirkpatrick, S., Gelatt, D., & Vecchi, M. (1983). Optimization by simulated annealing. science, 220, 671–680. 3, 33, 87, 119

Kittel, C. & Kroemer, H. (1980). Thermal physics. (Macmillan). 119

Kleinfeld, D. & Sompolinsky, H. (1988). Associative neural network model for the generation of temporal patterns. theory and application to central pattern generators. Biophysical journal, 54, 1039–1051. 12

Knight, B. W. (2000). Dynamics of encoding in neuron populations: some general mathematical features. Neural Computation, 12, 473–518. 17, 45, 46

Knill, D. & Kersten, D. (1991). Apparent surface curvature affects lightness perception. Nature, 351, 228–230. 55

Koch, C. (1999). Biophysics of Computation: Information Processing in Single Neurons. (Oxford University Press). 45

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (NIPS), F. Pereira, C. Burges, L. Bottou, & K. Weinberger, eds., pp. 1097–1105. (Curran Associates, Inc.). 6

Krzakala, F. & Kurchan, J. (2007). Landscape analysis of constraint satisfaction problems. Physical Review E, 76, 021122. 103

Lakatos, P., Karmos, G., Mehta, A., Ulbert, I., & Schroeder, C. (2008). Entrainment of neuronal oscillations as a mechanism of attentional selection. science, 320, 110–113. 41

Lee, S., Sen, K., & Kopell, N. (2009). Cortical Gamma Rhythms Modulate NMDAR-Mediated Spike Timing Dependent Plasticity in a Biophysical Model. PLoS Comput Biol, 5, e1000602+. 57

Linn, E., Rosezin, R., Tappertzhofen, S., Böttger, U., & Waser, R. (2012). Beyond von neumann - logic operations in passive crossbar arrays alongside memory operations. Nanotechnology, 23, 305205. 73

Lisman, J. & Spruston, N. (2010). Questions about stdp as a general model of synaptic plasticity. Frontiers in Synaptic Neuroscience, 2, 1–3. 74

Longtin, A. (1993). Stochastic resonance in neuron models. Journal of statistical physics, 70, 309–327. 107

Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. Proceedings of the IEEE, 102, 860–880. 87

Maass, W. & Markram, H. (2002). Synapses as dynamic memory buffers. Neural Networks, 15, 155–161. 76

MacKay, D. (2003). Information Theory, Inference and Learning Algorithms. (Cambridge University Press). 3, 87

Mamassian, P. & Goutcher, R. (2005). Temporal dynamics in bistable perception. Journal of Vision, 5, 7. 57, 59

Markov, N. & Kennedy, H. (2013). The importance of being hierarchical. Current opinion in neurobiology, 23, 187–194. 11

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. Science, 275, 213–215. 61

Markram, H., Toledo-Rodriguez, M., Wang, Y., Gupta, A., Silberberg, G., & Wu, C. (2004). Interneurons of the neocortical inhibitory system. Nature Reviews Neuroscience, 5, 793–807. 13, 45

Martin, A. & Nystrom, M. (2006). Asynchronous techniques for system-on-chip design. Proceedings of the IEEE, 94, 1089–1120. 70

Martin, K. (2011). Neuroanatomy: uninhibited connectivity in neocortex? Current Biology, 21, R425–R427. 13

Massimini, M., Ferrarelli, F., Huber, R., Esser, S., Singh, H., & Tononi, G. (2005). Breakdown of cortical effective connectivity during sleep. Science, 309, 2228–2232. 46

Mattia, M. & Del Giudice, P. (2002). Population dynamics of interacting spiking neurons. Physical Review E, 66, 051917. 17, 45, 46

Mayr, C. & Partzsch, J. (2010). Rate and pulse based plasticity governed by local synaptic state variables. Frontiers in Synaptic Neuroscience, 2, 28. 74, 76, 83

Mayr, C., Stärke, P., Partzsch, J., Cederstroem, L., Schüffny, R., Shuai, Y., Du, N., & Schmidt, H. (2012). Waveform driven plasticity in BiFeO3 memristive devices: Model and implementation. In Advances in Neural Information Processing Systems 25, pp. 1700–1708. 74

McCormick, D., Connors, B., Lighthall, J., & Prince, D. (1985). Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex. Journal of neurophysiology, 54, 782–806. 13

McCulloch, W. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys., 5, 115–133. 2, 25

Mead, C. (1990). Neuromorphic electronic systems. Proceedings of the IEEE, 78, 1629–36. 34

Merolla, P., Arthur, J., Alvarez, R., Bussat, J.-M., & Boahen, K. (2014a). A multicast tree router for multichip neuromorphic systems. Circuits and Systems I: Regular Papers, IEEE Transactions on, 61, 820–833. 104

Merolla, P., Arthur, J., Shi, B., & Boahen, K. (2007). Expandable networks for neuromorphic chips. IEEE Transactions on Circuits and Systems I, 54, 301–311. 66

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R., & Modha, D. S. (2014b). A million spiking-neuron integrated circuit with a scalable communication network and interface. Science, 345, 668–673. 7, 73, 107

Mézard, M., Parisi, G., & Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. Science, 297, 812–815. 103

Miltner, W., Braun, C., Arnold, M., Witte, H., & Taub, E. (1999). Coherence of gamma-band eeg activity as a basis for associative learning. Nature, 397, 434–436. 61

Minsky, M. & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. (MIT Press, Cambridge, Mass). 59, 87

Mitra, S., Fusi, S., & Indiveri, G. (2009). Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. Biomedical Circuits and Systems, IEEE Transactions on, 3, 32–42. 83

Moreno, C., Munuera, C., Valencia, S., Kronast, F., Obradors, X., & Ocal, C. (2010). Reversible resistive switching and multilevel recording in La0.7Sr0.3MnO3 thin films for low cost nonvolatile memories. Nano letters, 10, 3828–3835. 73

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient sat solver. In Proceedings of the 38th annual Design Automation Conference, pp. 530–535. ACM. 92

Mostafa, H., Corradi, F., Osswald, M., & Indiveri, G. (2013a). Automated synthesis of asynchronous event-based interfaces for neuromorphic systems. In Circuit Theory and Design, (ECCTD) 2013 European Conference on, pp. 1–4. (IEEE). ix

Mostafa, H., Corradi, F., Stefanini, F., & Indiveri, G. (2014). A hybrid analog/digital spike-timing dependent plasticity learning circuit for neuromorphic VLSI multi-neuron architectures. In International Symposium on Circuits and Systems, (ISCAS), 2014, pp. 854–857. (IEEE). 66

Mostafa, H. & Indiveri, G. (2014). Sequential activity in asymmetrically coupled winner-take-all circuits. Neural computation, 26, 1973–2004. ix

Mostafa, H., Khiat, A., Serb, A., Mayr, C., Indiveri, G., & Prodromakis, T. (2015a). Implementation of a spike-based perceptron learning rule using tio2- x memristors. Frontiers in Neuroscience. ix

Mostafa, H., Müller, L. K., & Indiveri, G. (2013b). Recurrent networks of coupled winner-take-all oscillators for solving constraint satisfaction problems. In Advances in Neural Information Processing Systems (NIPS), C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger, eds., vol. 26, pp. 719–727. ix

Mostafa, H., Müller, L. K., & Indiveri, G. (2015b). An event-based architecture for solving constraint satisfaction problems. Nature Communications. (In press). ix, 91

Mostafa, H., Müller, L. K., & Indiveri, G. (2015c). Rhythmic inhibition allows neural networks to search for maximally consistent states. Neural computation. (in press). ix, 57, 108

Muller, D. E. & Bartky, W. (1959). A theory of asynchronous circuits. In Proceedings of an International Symposium on the Theory of Switching, vol. 29, pp. 204–243. 67

Muller, L. (2015). Algorithms for massively parallel, event-based hardware. Ph.D. thesis, University of Zurich. 108

Nagamatu, M. & Yanaru, T. (1996). On the stability of lagrange programming neural networks for satisfiability problems of prepositional calculus. Neurocomputing, 13, 119–133. 87

Navaridas, J., Furber, S., Garside, J., Jin, X., Khan, M., Lester, D., Luján, M., Miguel-Alonso, J., Painkras, E., Patterson, C., et al. (2013). SpiNNaker: Fault tolerance in a power-and area-constrained large-scale neuromimetic architecture. Parallel Computing, 39, 693–708. 7

Neftci, E., Binas, J., Rutishauser, U., Chicca, E., Indiveri, G., & Douglas, R. (2013). Synthesizing cognition in neuromorphic electronic systems. Proceedings of the National Academy of Sciences, 110, E3468–E3476. 25, 30, 31

Nessler, B., Pfeiffer, M., & Maass, W. (2013). Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. PLoS computational biology, 9, e1003037. 6

Newell, A. & Simon, H. (1976). Computer science as empirical inquiry: Symbols and search. Communications of the ACM, 19, 113–126. 2, 25

Noack, M., Partzsch, J., Mayr, C., Hänzsche, S., Scholze, S., Höppner, S., Ellguth, G., & Schüffny, R. (2015). Switched-capacitor realization of presynaptic short-term-plasticity and stop-learning synapses in 28 nm CMOS. Frontiers in Neuroscience, 9. 73, 83

Papadimitriou, C. (1991). On selecting a satisfying truth assignment. In Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on, pp. 163–169. IEEE.

Park, J., Ha, S., Yu, T., Neftci, E., & Cauwenberghs, G. (2014). A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver. In Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE, pp. 675–678. IEEE. 7

Penrose, R. (1999). The emperor's new mind: Concerning computers, minds, and the laws of physics. (Oxford University Press). 1

Pfister, J.-P., Toyoizumi, T., Barber, D., & Gerstner, W. (2006). Optimal spike-timing dependent plasticity for precise action potential firing in supervised learning. Neural Computation, 18, 1309–1339. 74, 76, 83

Pham, P.-H., Jelaca, D., Farabet, C., Martini, B., LeCun, Y., & Culurciello, E. (2012). Neuflow: Dataflow vision processing system-on-a-chip. In Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on, pp. 1044–1047. 107

Piccinini, G. & Scarantino, A. (2011). Information processing, computation, and cognition. Journal of biological physics, 37, 1–38. 2

Plesser, H. & Gerstner, W. (2000). Noise in integrate-and-fire neurons: From stochastic input to escape rates. Neural Computation, 12, 367–384. 112

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., & Indiveri, G. (2015). A re-configurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. Frontiers in Neuroscience, 9. 7, 66, 75, 76, 79

Rabinovich, M., Huerta, R., Varona, P., & Afraimovich, V. (2008). Transient cognitive dynamics, metastability, and decision making. PLoS Comput. Biol, 4, e1000072. 12, 31

Rabinovich, M., Volkovskii, A., Lecanda, P., Huerta, R., Abarbanel, H., & Laurent, G. (2001). Dynamical encoding by networks of competing neuron groups: winnerless competition. Physical Review Letters, 87, 068102. 12

Ramsey, R., Cross, E., & Hamilton, A. (2013). Supramodal and modality-sensitive representations of perceived action categories in the human brain. Experimental brain research, 230, 345–357. 11

Ray, S. & Maunsell, J. (2010). Differences in gamma frequencies across visual cortex restrict their possible use in computation. Neuron, 67, 885–896. 45, 120

Roberts, M., Lowet, E., Brunet, N., Wal, M. T., Tiesinga, P., Fries, P., & Weerd, P. D. (2013). Robust gamma coherence between macaque v1 and v2 by dynamic frequency matching. Neuron, 78, 523–536. 52

Rotter, S. & Aertsen, A. (1998). Accurate spike synchronization in cortex. Z Naturforsch C, 53, 686–90. 31

Ruiz, I. R. & Romay, M. G. (2011). Gravitational swarm approach for graph coloring. In Nature Inspired Cooperative Strategies for Optimization (NICSO 2011). (Springer), pp. 159–168. 95, 96

Rumelhart, D. & McClelland, J. (1986). Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations. (Cambridge, MA, USA: MIT Press). 2, 59, 87, 117

Rumelhart, D. E. (1998). The architecture of mind: A connectionist approach. Mind readings, pp. 207–238. 2, 117

Rutishauser, U. & Douglas, R. (2009). State-dependent computation using coupled recurrent networks. Neural Computation, 21, 478–509. 13, 25, 30, 31

Rutishauser, U., Douglas, R., & Slotine, J. (2011). Collective stability of networks of winner-take-all circuits. Neural computation, 23, 735–773. 13, 34

Rutishauser, U., Slotine, J., & Douglas, R. (2012). Competition through selective inhibitory synchrony. Neural Computation, 24, 2033–2052. 13, 44, 60

Saighi, S., Mayr, C., Linares-Barranco, B., Serrano-Gotarredona, T., Schmidt, H., Lecerf, G., Tomas, J., Grollier, J., Boyn, S., Alibart, F., La Barbera, S., Vuillaume, D., Bichler, O., Gamrat, C., Vincent, A., & Querlioz, D. (2015). Plasticity in memristive devices. Frontiers in Neuroscience, 9. 73

Salakhutdinov, R. & Hinton, G. (2009). Deep boltzmann machines. In International Conference on Artificial Intelligence and Statistics, pp. 448–455. 107

Schemmel, J., Brüderle, D., Meier, K., & Ostendorf, B. (2007). Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In International Symposium on Circuits and Systems, (ISCAS), 2007, pp. 3367–3370. IEEE. 73

Schemmel, J., Grübl, A., Hartmann, S., Kononov, A., Mayr, C., Meier, K., Millner, S., Partzsch, J., Schiefer, S., Scholze, S., Schüffny, R., & Schwartz, M. (2012). Live demonstration: A scaled-down version of the BrainScaleS wafer-scale neuromorphic system. In IEEE International Symposium on Circuits and Systems ISCAS 2012, p. 702. 73

Senn, W. & Fusi, S. (2005). Learning Only When Necessary: Better Memories of Correlated Patterns in Networks with Bounded Synapses. Neural Computation, 17, 2106–2138. 74

Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., & Linares-Barranco, B. (2013). STDP and STDP variations with memristors for spiking neuromorphic learning systems. Frontiers in Neuroscience, 7. 74

Shor, P. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM journal on computing, 26, 1484–1509. 4

Shuai, Y., Ou, X., Luo, W., Du, N., Wu, C., Zhang, W., Burger, D., Mayr, C., Schüffny, R., Zhou, S., Helm, M., & Schmidt, H. (2013). Nonvolatile multilevel resistive switching in Ar+ irradiated BiFeO3 thin films. IEEE Electron Device Letters, 34, 54–56. 73

Silva, J. & Sakallah, K. (1997). Grasp a new search algorithm for satisfiability. In Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, pp. 220–227. IEEE Computer Society. 92

Sipser, M. (1996). Introduction to the Theory of Computation. (International Thomson Publishing). 7, 90, 118

Sjöström, P., Rancz, E., Roth, A., & Häusser, M. (2008). Dendritic excitability and synaptic plasticity. Physiological Reviews, 88, 769–840. 74

Sjöström, P., Turrigiano, G., & Nelson, S. (2001). Rate, Timing, and Cooperativity Jointly Determine Cortical Synaptic Plasticity. Neuron, 32, 1149–1164. 21, 35, 57, 74, 76

Smith, K., Palaniswami, M., & Krishnamoorthy, M. (1998). Neural techniques for combinatorial optimization with applications. Neural Networks, IEEE Transactions on, 9, 1301–1318. 34

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. 2, 107

Sohal, V. S., Zhang, F., Yizhar, O., & Deisseroth, K. (2009). Parvalbumin neurons and gamma rhythms enhance cortical circuit performance. Nature, 459, 698–702. 45

Soltani, A. & Wang, X.-J. (2006). A biophysically based neural model of matching law behavior: melioration by stochastic synapses. The Journal of neuroscience : the official journal of the Society for Neuroscience, 26, 3731–3744. 31

Stein, A. V. & Sarnthein, J. (2000). Different frequencies for different scales of cortical integration: from local gamma to long range alpha/theta synchronization. International Journal of Psychophysiology, 38, 301–313. 43, 44

Strogatz, S. (2000). From kuramoto to crawford: exploring the onset of synchronization in populations of coupled oscillators. Physica D: Nonlinear Phenomena, 143, 1–20. 52, 134

Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. nature, 453, 80–83. 73

Sundareswara, R. & Schrater, P. (2008). Perceptual multistability predicted by search model for bayesian decisions. Journal of Vision, 8, 12. 52

Tallon-Baudry, C., Bertrand, O., Delpuech, C., & Permier, J. (1997). Oscillatory gamma-band (30–70 hz) activity induced by a visual search task in humans. J. Neurosci., 17, 722–734. 43

Thomson, A., Bannister, A., Mercer, A., & Morris, O. (2002a). Target and temporal pattern selection at neocortical synapses. Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 357, 1781–1791. 12, 14

Thomson, A., West, D., Wang, Y., & Bannister, A. (2002b). Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2–5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. Cerebral cortex, 12, 936–953. 13

Tiesinga, P. & Sejnowski, T. (2009). Cortical enlightenment: are attentional gamma oscillations driven by ing or ping? Neuron, 63, 727–732. 45, 60

Van Essen, D., Anderson, C., & Felleman, D. (1992). Information processing in the primate visual system - An integrated systems perspective. Science, 255, 419–423. 11

Verduzco-Flores, S., Bodner, M., & Ermentrout, B. (2012). A model for complex sequence learning and reproduction in neural populations. Journal of computational neuroscience, 32, 403–423. 12

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. The Journal of Machine Learning Research, 11, 3371–3408. 107

von Helmholtz, H. & Southall, J. (1925). Treatise on physiological optics. III. The perceptions of vision. (New York: Optical Society of America). 2, 4, 44

Wang, X. (2002). Probabilistic decision making by slow reverberation in cortical circuits. Neuron, 36, 955–968. 31, 129

Wang, X. (2008). Decision making in recurrent neuronal circuits. Neuron, 60, 215–234. 31, 41

Wang, X. (2010). Neurophysiological and computational principles of cortical rhythms in cognition. Physiological reviews, 90, 1195–1268. 40

Watts, J. & Thomson, A. (2005). Excitatory and inhibitory connections show selectivity in the neocortex. The Journal of Physiology, 562, 89+. 12, 13, 14

Weiss, S. & Rappelsberger, P. (2000). Long-range eeg synchronization during word encoding correlates with successful memory performance. Cognitive Brain Research, 9, 299–312. 61

Whittington, M., Cunningham, M., LeBeau, F., Racca, C., & Traub, R. (2011). Multiple origins of the cortical gamma rhythm. Developmental neurobiology, 71, 92–106. 60

Womelsdorf, T., Schoffelen, J.-M., Oostenveld, R., Singer, W., Desimone, R., Engel, A., & Fries, P. (2007). Modulation of neuronal interactions through neuronal synchronization. science, 316, 1609–1612. 40, 43, 45, 46

You, T., Shuai, Y., Luo, W., Du, N., Bürger, D., Skorupa, I., Hübner, R., Henker, S., Mayr, C., Schüffny, R., Mikolajick, T., Schmidt, O., & Schmidt, H. (2014). Exploiting memristive BiFeO3 bilayer structures for compact sequential logics. Advanced Functional Materials, 24, 3357–3365. 73

Zhang, S. & Constantinides, A. (1992). Lagrange programming neural networks. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, 39, 441–452. 87